September, 1984

CK-LOG, A CALCULUS FOR KNOWLEDGE PROCESSING IN LOGIC

by

C.V. Srinivasan

DCS-TR-153

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

This work was supported by ONR/ONT program element 62721N.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **SEP 1984**	2. REPORT TYPE	3. DATES COVERED **00-00-1984 to 00-00-1984**

4. TITLE AND SUBTITLE **CK-LOG, A Calculus for Knowledge Processing in Logic**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Rutgers University,Department of Computer Science,New Brunswick,NJ,08903**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**111**	

ACKNOWLEDGEMENT

I am thankful to the Navy Center for Applied Research in Artificial Intelligence for supporting this work since summer of 1981.

CK-LOG

A CALCULUS FOR KNOWLEDGE PROCESSING IN LOGIC

Chitoor V. Srinivasan

Table of Contents

CK-LOG

A CALCULUS FOR KNOWLEDGE PROCESSING IN LOGIC.

Chitoor V. Srinivasan
Navy Center for Applied Research in Artificial Intelligence,
Code 7510, Naval Research Laboratory, Washington, D.C. 20375
and
Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903.
September, 6 1984.

ABSTRACT

This paper introduces the principal concepts in the organization and operation of the logic based knowledge processing system, called CK-LOG (A Calculus for Knowledge in LOGic). CK-LOG uses the *frame* based system MDS (the Meta Description System) for knowledge representation and for modelling world states. It uses an inference engine based on *Natural Deduction* for stating and solving problems.

As a knowledge processing system CK-LOG has several capabilities which are new to the technology of knowledge representation systems: CK-LOG has special facilities to represent and reason about actions and their time dependencies. Actions that occur in a world state may create or destroy objects in the world or modify their properties, or prevent or support other actions. The effects of actions are described in CK-LOG using *modal operators* like CREATE, DESTROY, PREVENT, SUPPORT, KEEP, etc. These operator expressions are also used to represent and reason about possible worlds that the actions might lead to.

Most significantly, CK-LOG is a logic based knowledge processing system, just as PROLOG is a logic based programming system. CK-LOG uses a three valued logical system with truth values T (true), ? (unknown) and F (false) to build partial models of world states, and the two valued logical system of T and F in its theorem proving system. The use of the three valued logical system in its models of world states enables CK-LOG to do problem solving in the context of incomplete information about world states.

The theorem proving system of CK-LOG uses a variant of the *calculus of sequents* first proposed by Kanger (which itself is a variant of Gentzen's system). The two variations in CK-LOG are, (i). the use of a new algorithm called the *mating algorithm* for testing proof terminations, and (ii) the use of specialized inference rules for reasoning about *modal expressions* using the *possible world* semantics.. The mating algorithm gives the theorem proving system of CK-LOG several new capabilities: to identify information that is pertinent to a given problem and retrieve it from its knowledge base, to update its models of possible worlds during the problem solving process based on the findings of the theorem proving system, to use these models of world states to test proof terminations, and to generate hypotheses during the problem solving process that are based on unknown information.

These various features of CK-LOG are described here. The paper concludes with a discussion of the logic of *frames* as used in CK-LOG and establishes a condition called *locality condition* as a sufficient condition for creating knowledge representations with requisite completeness.

CK-LOG,

A CALCULUS FOR KNOWLEDGE PROCESSING IN LOGIC*

Chitoor V. Srinivasan†

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory - Code 7510, Washington, D.C. 20375
and
Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903.
September 1984.

1. Introduction.

Hierarchical representations based on *frames* [Minsky 1975] have been extensively used in AI systems for *knowledge representation* and *problem solving* [Goldstein 1976] [Bobrow 1977] [Sridharan 1978] [Stefik 1979] [Schmolze 1982]. The *frame definitions* are used in such AI systems to define classes of objects in a problem domain and elements of a *domain language*, DL, to describe situations in the *world states* of the domain. *Models* of world states described in DL are created through *frame instantiations*. *Problems* are stated to such systems as specifications in DL of models to be built or modified. The controlling principle for problem solution is provided by the notion of *model consistency*: A model generated by the system is a solution to a problem only if it *satisfies* the given specification and is *contradiction free*. *Frame* based AI systems have demonstrated significant problem solving competence in several experimental domains [Goldstein 1976] [Bobrow 1977] [Stefik 1979] [Lenat 1976]. It seems appropriate that in the context of this kind of competence the *frame based systems* acquired the special identity as *knowledge representation systems* rather than just new programming systems.

The *knowledge engineer* who defines the *frames* is required to define also the procedures for *frame instantiations*, and procedures for keeping a set of such instantiations contradiction free. *Frames* provide a convenient way of organizing the domain information into *data structures* (symbol structures) over which the knowledge engineer could write the needed problem solving programs. The interpretation given to *frames* by these programs will define the semantics of *frames* in these systems. *Frames* as units of

* This work was supported by ONR/ONT Grant 62721N. † Currently on sabbatical leave from Rutgers.

knowledge do not thus have a standard logical semantics. This has had two undesirable consequences:

(i). It has led to a proliferation of varieties of *frame based* knowledge representation systems. Each of these systems established its usefulness in a limited way in the context of the particular applications that inspired their design. But they have failed to advance our scientific understanding of the design and implementation of such systems. I do not think that generation of yet more *frame based knowledge representation systems* would significantly change this state of affairs.

(ii). It imposes on the user of any of these systems an enormous programming overhead with its usual problems: lack of flexibility for growth and modifications, difficulty of maintenance, and difficulties associated with *technology transfer* to new domains of applications. This seems contrary to what should happen, because if one knew how to represent and process knowledge in a computer, it seems it should make computers easier to use and new systems easier to develop.

This impasse is not due to the organizational principle provided by *frames*, which the above systems exploited, but it is because, (a). we did not have the logical foundational knowledge that was needed to characterize the nature of these systems and their design, and (b). we did not have the technical means to integrate the principle of problem solving through *theorem proving* with *frame based* system organizations. The work of Levesque [Levesque 1984] now gives us the logical foundations we needed to understand the nature and function of these systems. The work on CK-LOG (a Calculus for Knowledge in LOGic) presented here integrates *frame* based knowledge representation with *theorem proving*. CK-LOG may be viewed as being an instantiation of the knowledge representation system characterized by Levesque**.

CK-LOG is a logic based knowledge processing system, just as PROLOG is a logic based programming system. It uses the *frame based* system MDS (the Meta Description System) [Srinivasan 1973, '77] for knowledge representation and modeling *world states*, and a theorem proving system, the *inference engine* of CK-LOG that is based on *natural deduction*, for stating and solving problems. CK-LOG's inference engine uses a variant of the *calculus of sequents* first proposed by Gentzen [Gentzen 1935] and later modified by [Kanger 1963]. The variant used in CK-LOG employs a new algorithm called the

** My understanding of CK-LOG and the writing of this paper have both been considerably influenced by Levesque's work. CK-LOG is now being used to implement a consultation system, called OPPLAN-CONSULTANT, for Naval Operational Planning [Srinivasan 1984].

mating algorithm for testing proof terminations, and uses specialized inference rules for reasoning about *modal expressions*. Modal expressions are used in CK-LOG to represent and reason about ongoing actions, their time dependencies and the *possible worlds* that they might lead to, and to mediate communication between the inference engine, the *knowledge base* and *models* of world states. Models are used in CK-LOG both to check the consistency of the inference steps in its theorem proving process, and to identify and retrieve *domain knowledge* that is pertinent to the problem being solved. Problems are stated in CK-LOG as *sequents* which are generalizations of the *Horn clause* forms used in PROLOG. A problem represented by a sequent is the analog of Levesque's ASK operator. A problem is said to be solved when the sequent representing the problem is proven *valid*.

CK-LOG uses MDS in two ways: (i) to define and represent domain knowledge and model world states, and (ii) to define and represent inference rules and model *problem states*. Thus, the CK-LOG architecture gives it, in principle, the full meta-level capabilities to reason about its own operations*.

Actions in CK-LOG may create, or destroy objects in a world or modify their properties, or prevent or support other actions. Operators like CREATE, DESTROY, PREVENT, SUPPORT, KEEP, ASSERT, etc., are used in the logical language of CK-LOG as *modal operators*. The modal expressions using these operators are interpreted in CK-LOG in two ways, (i) as implicitly referring to the actions that are needed to perform the indicated operations, and (ii) as describing situations in the possible worlds that the actions might lead to. The ASSERT-expressions mediate communication between the inference engine and models of world states. These are the analogs of Levesque's TELL operator. A special inference rule, called **KB**-lookup (Knowledge Base lookup), is used to mediate communication between the inference engine and the knowledge base. The *modal operators* ISTRUE, ISFALSE and ISUNKNOWN that are used in the meta-language of CK-LOG correspond to the **K** operator used by Levesque.

The modeling system MDS, that is used in CK-LOG, uses the three valued logic, T (True), F (False) and ? (Unknown), T > ? > F, and (NOT ?) = ?. It models *actions* and

* I do not think the current implementation of CK-LOG would allow this. The implementation of the theorem proving system of CK-LOG is now in progress.

time dependencies among the *events* over a lattice of *time instants*, t. Time intervals are represented both by open intervals, $(t_1\ t_2)$ as in (between $(t_1\ t_2)$), or semi-closed intervals $[t_1\ t_2)$ as in (during $[t_1\ t_2)$). MDS is implemented in Rutgers ELISP on the DEC-TOPS-20 system. The logic of MDS and CK-LOG are described in this paper. The way we are now using CK-LOG to implement the *expert consultant*, OPPLAN-CONSULTANT, for Naval Operational Planning is discussed in [Srinivasan 1984].

The architecture of CK-LOG is shown schematically by the block diagram in figure 1.1. The two principal components of the system are represented in this diagram by the two L-shaped regions with bold outlines. The outer L-region is MDS consisting of its three components, the *knowledge representation system*, KRS, the *knowledge base*, KB, and the *truth maintenance system*, TMS, associated with the *world states*. The inner L-region represents the *theorem proving system*, TPS, consisting of its three components, *meta-system for defining inference rules*, MSI, the knowledge base of *inference rules*, KBI, and the *inference engine*, IE, with its associated *problem states*. MSI is implemented using KRS. The rules in KBI are represented using structures similar to those in KB. IE is partly implemented using the facilities available in TMS, and partly through specialized routines written in ELISP. The block between the two L's, called UI, is the *user interface*.

This paper is organized in four major sections. Section 2 introduces the calculus of sequents, the theorem proving system TPS, the *mating algorithm* and the modifications to the calculus of sequents introduced in CK-LOG to communicate with models of world states and the knowledge base, as well as to reason about the possible worlds represented by *modal expressions*. Section 3 presents the principal modeling facilities of MDS. The way MDS is being used to implement TPS is briefly outlined in Appendix I. Section 4 presents an example that illustrates the use of TPS and TMS to analyze, plan and execute actions in CK-LOG. Section 5 presents the logic of *frames* as they have been used in MDS and introduces the *locality principle* that is used as a guiding principle for defining knowledge. The paper concludes with a summary of the principal contributions made by CK-LOG to the technology of knowledge representation systems. The knowledge description paradigm used in CK-LOG is outlined in the next subsection.

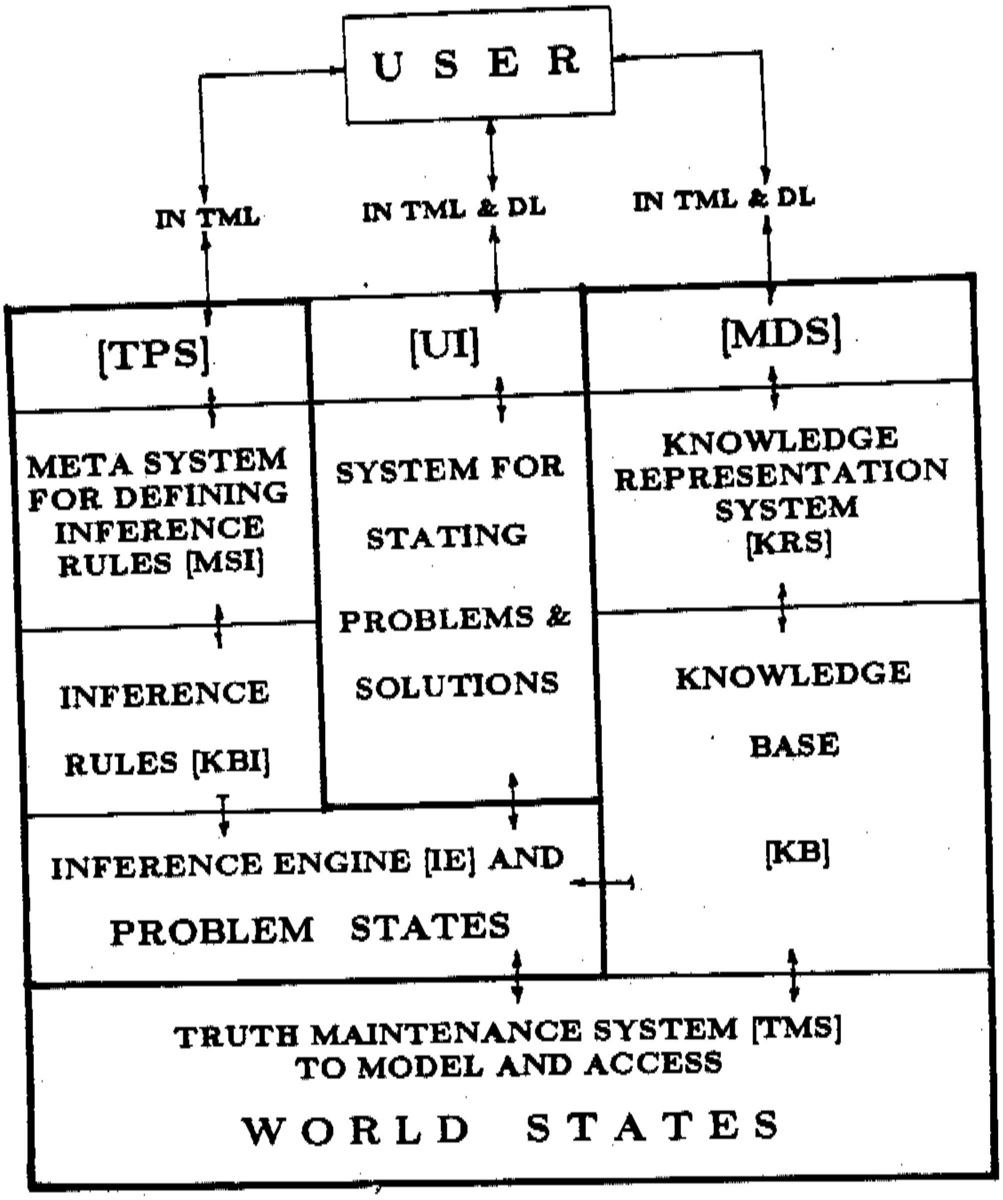

Figure 1.1: The Architecture of CK-LOG.

1.1. The Knowledge Description Paradigm.

We begin with a universe U. This universe consists of physical objects, actions and concepts. Concepts are sets of objects and/or actions. Actions modify the objects, concepts and actions themselves. The state of this universe at time, t, is called the *world* *state*†, denoted by $\mathbf{U}_t$.

Let X be a concept in **U**. It is assumed that one's understanding of X is determined by one's knowledge of the five aspects of X shown below:

structure: Components of X, i.e. the set of elements of U that together constitute X, and the way they are related to X.

function: What X is intended for. Statements which can be made true by X in the universe U.

behavior: Valid statements about the modifications in U that may occur while X is performing its function.

analysis: Generating valid statements about the *function*, (or *function* and *behavior*) of X from given specifications of *structure* (or *structure* and *function*) of X.

design: This is the converse of analysis; generating valid statements about *structure* (or *structure* and *function*) of X, from given specifications of *function* (or *behavior* and *function*) of X.

The language used to describe these five aspects is called **TML** (The Meta Language). For a concept X in **U**, let **K**(X) denote the *knowledge of* X, namely the definition of the above five aspects of X in the language **TML**. Let **K**[**U**] be the *knowledge of* **U**, namely the set of all **K**(X) for X in **U**. (CK-LOG + **K**[**U**]) is the knowledge based problem solving system for the universe **U**.

As we shall see below, definitions of *structures* of concepts in **U** will introduce to CK-LOG the elements of a logical language, **DL** (The Domain Language). This is the language used by CK-LOG to describe situations in the world states $\mathbf{U}_t$. **DL** is the sublanguage of **TML** without modal expressions, that is specialized to the given universe, **U**.

Another part of the definition of the knowledge of a given universe in CK-LOG is the definition of the *operators* that are used in **TML** to describe situations in possible future worlds and to communicate with the knowledge base and world states. Operator expressions like**

† Throughout this paper I will use the phrase 'world state' for 'models of world state'. Occasionally I will be referring to the 'world states' that models represent. But this will not result in any confusion.

** Example taken from OPPLAN-CONSULTANT domain.

[CREATE
((EXISTS $region_1$ (southern-shore-of France))
((during $interval_1$)(controlled-by $region_1$ Allies)))]

are interpreted by CK-LOG as referring to the truth or falsity of their arguments in *hypothetical worlds*, the worlds in which the actions, that are implicitly denoted by the expressions, have terminated. For a given arbitrary logical expression, CK-LOG uses its knowledge base and inferencing facilities to identify the actions that are associated with it by a given operator. Thus, the statement above specifies that in some possible future world, because of the successful completion of the actions implicitly referred to by the CREATE operator, it might be true that the Allies have control of some region in the southern-shore-of France, during the specified interval.

If this statement is given to CK-LOG as a planning goal, then CK-LOG will use its inference engine, its model of the current world state and its knowledge base to identify the actions that are needed to select a region in southern France and take control of it, and analyze the actions (and the possible enemy actions) in order to plan for them. Thus, statements in CK-LOG have both declarative and procedural interpretations.

Analysis methods for analyzing expressions like these are defined to CK-LOG by specifying inference rules for them. The inference rules are used by the inference engine to identify and create the actions needed to make the expressions true. Thus, **TML** is used not only to define the knowledge **K[U]**, but also to define the domain language **DL** and inference processes for the modal operators. Hence the name 'The Meta Language'.

Using a logical language as the domain language gives us two advantages: Descriptions of **K[U]** in **TML** and **DL** have well defined logical semantics, and one can use general logical deductive techniques to state and solve problems in the universe **U**, without having to write specialized programs to tell CK-LOG how it should use **K[U]** in its problem solving processes. CK-LOG does not attempt to solve difficult problems by itself. It acts as an intelligent assistant to a human problem solver and uses its models of world states to guide itself in its problem solving processes. The way this is done in CK-LOG is described in the ensuing sections.

2. TPS, the Theorem Proving System of CK-LOG.

2.1. Sequents, Problems, and Theorems.

A *sequent* has the form,

$$[S1]:\ X_1, ..., X_n \rightarrow Y_1, ..., Y_m\ ;\ \ 0 \leq n, \text{ and } 0 \leq m.$$

where the X's and the Y's are arbitrary logical expressions in **TML**. One may read this sequent as,

From $(X_1$ and X_2 and ... and $X_n)$ conclude $(Y_1$ or Y_2 or ... or $Y_m)$.

The logical expressions on the left side of → may be viewed as known facts, or hypotheses. The sequent says, given that the conjunction of known facts or hypotheses is true, one may conclude the disjunction of the expressions on the right side of →.

A sequent is said to be *valid* iff

$$[L1]:\ [(X_1 \text{ AND } ... \text{ AND } X_n) \text{ IMPLIES } (Y_1 \text{ OR } ... \text{ OR } Y_m)],$$

is a *theorem*. This means that in every model (world state) in which the antecedent is true the consequent is also true. One may think of this as saying that a sequent is valid iff the conclusion proposed in the sequent is provably correct. A bullet is placed instead of ';' in a sequent to indicate that the sequent is valid as in,

$$[T1]:\ X_1, ..., X_n \rightarrow Y_1, ..., Y_m \bullet$$

A sequent without this bullet is interpreted as a problem to be solved. The problem is, of course, to prove that the sequent is valid. The problem,

$$[P1]:\ \rightarrow Y_1, ..., Y_m\ ;$$

with empty left side is interpreted in CK-LOG as,

Given **K[U]** and the current $\mathbf{U}_t$ prove that
one may conclude (Y_1 OR ... OR Y_m).

The problem,

$$[P2]: \quad X_1, ..., X_n \rightarrow Y_1, ..., Y_m ;$$

is interpreted as,

Assuming the hypotheses X_1, ... and X_n, and given **K[U]** and $\mathbf{U}_t$, prove that one may conclude (Y_1 OR ... OR Y_m).

Thus in CK-LOG the logical statements corresponding to **K[U]** and the facts in $\mathbf{U}_t$ are assumed to be always available on the left side of each problem. When a user types

→ (ACHIEVE (*design* opplan)) ;

to CK-LOG, the user is stating a problem to CK-LOG. The problem here is to achieve the design of the operational plan, opplan. The specifications for the design of the opplan are assumed to be already in the current world state. CK-LOG will use its inference engine and its model building capability to interpret the above problem, to identify the knowledge in **K[U]** that is relevant to the problem, translate it to the appropriate logical expressions in **DL**, place them in the problem sequent when needed, and proceed in its attempt to solve the problem (details in [Srinivasan 1984]). To understand this process it is first necessary to know what the concept of a proof is in the language of sequents, and how proofs are constructed. This is discussed in the next subsection.

2.2. The Concept of a Proof.

In presenting the concept of a proof and the calculus of sequents, to simplify the discussion, I will assume that **K[U]** and $\mathbf{U}_t$ are empty, and for each problem all the needed facts and hypotheses are explicitly given on the left side of the problem sequent. The calculus of sequents assumes a single axiom. The axiom is,

$$X \rightarrow X \quad \bullet$$

namely that 'from X one may conclude X' for any logical expression X. The general statement of this axiom has the form,

[A]: .., X, ... → ...., X, ... •

where the dots indicate that the expression X may be surrounded on both the left and the right side of → by an arbitrary number of (zero or more) other logical expressions. The differing number of dots indicate that the expressions in the various locations need not all be the same. Clearly,

[(.. AND X AND ...) IMPLIES (.... OR X OR .....)]

is a theorem. Thus axiom [A] is valid.

The calculus of sequents provides rules for transforming a problem sequent to one or more simpler problem sequents, preserving the validity of the sequent in this process, i.e. if the original sequent is valid then all the transformed sequents are also valid and if the original sequent is not valid then at least one of the transformed sequents is not valid. Through successive transformations if one is able to reduce a problem sequent to the form of the axiom [A] (or to a set of sequents each having the form of [A]) then, since all transformations preserve validity, the problem sequent should also be valid and this would constitute the proof of the problem. This process is illustrated in figure 2.1, where '[A]' in the leaves of the tree indicate that the sequent at that place is an axiom, and '[P]' indicates that the sequent at that place is a problem. If one is not able to reduce the problem sequent at the root of this tree to axioms in this manner then the validty of the problem sequent is not proven. The tree of deductions shown in figure 2.1 is called *proof tree* or *deduction tree*. The set of all leaf sequents in a deduction tree is called the *frontier set* of the tree.

The set of rules used in the calculus is *complete* in the sense that if one started with a valid sequent at the root then one is guaranteed termination with axioms in all leaf nodes of the proof tree. It is *consistent* in the sense that if one found a proof tree with axioms in all leaf nodes then the problem at the root is indeed valid. I will present this calculus in four stages: In the first stage I will introduce the rules for proving

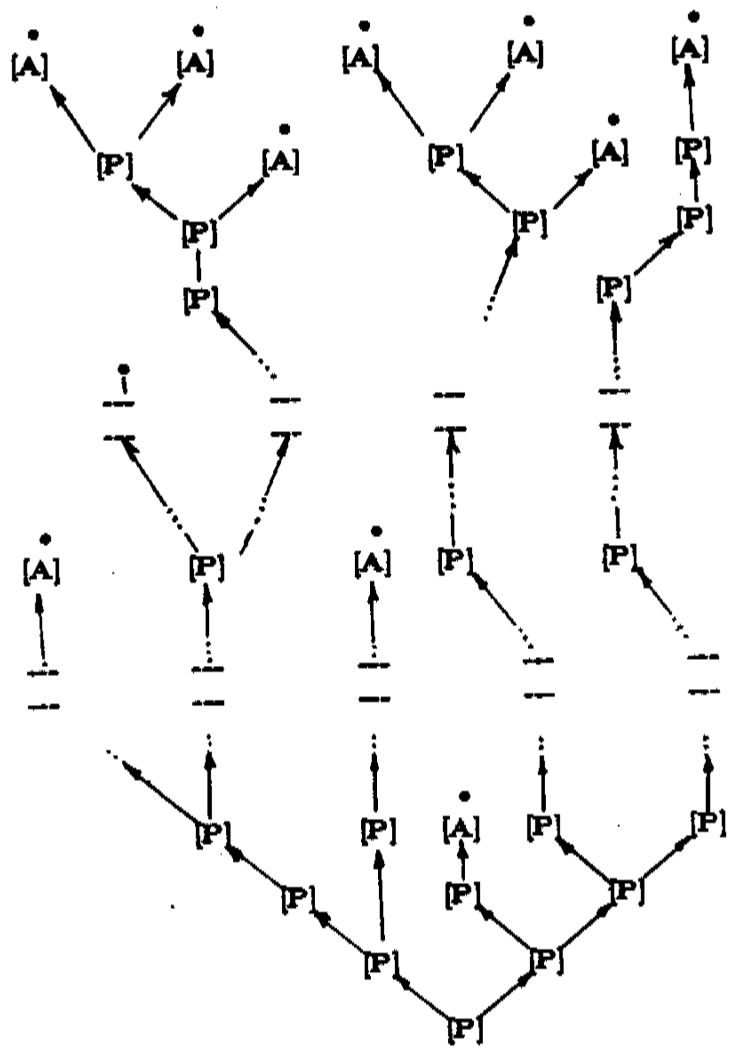

Figure 2.1: The Structure of a Proof Tree in the Calculus of Sequents.

propositional expressions (i.e. expressions that do not contain quantifiers, EVERY or EXISTS, or operators) and illustrate the use of the rules with a simple example. In the second stage I will introduce the rules for reducing quantified expressions and again illustrate their use with an example. In the third stage I will introduce the modifications to this calculus that have been incorporated in CK-LOG to use **K[U]** and the world state.

In the fourth stage, I will introduce the rules for reducing modal expressions and timed expressions. The *action calculus* defined by these rules is then illustrated in section 4 through a simple example.

2.3. The Propositional Calculus of Sequents.

The transformation rules are shown in table 2-I. Each rule has two parts: a bottom sequent and a top sequent. It is to be interpreted as follows: Any time a problem sequent is found that matches the pattern shown in the bottom sequent, it may be transformed to the sequent(s) shown on the top. The dots again indicate that the patterns shown may be surrounded by an arbitrary number of other expressions in the sequent, separated by commas.

TABLE 2-I: PROPOSITIONAL RULES.

Rule Name	Inference
[AND →]	...., x, y, .. → .... ;
	...., (x AND y), .. → .... ;
[→ AND]	.... → ...., x, .. ; | .... → ...., y, .. ;
	.... → ...., (x AND y), .. ;
[→ OR]	.... → ...., x, y, .. ;
	.... → ...., (x OR y), .. ;
[OR →]	...., x, .. → .... ; | ...., y, .. → .... ;
	...., (x OR y), .. → .... ;
[→ N]	...., x → ...., .. ;
	.... → ...., (NOT x), .. ;
[N →]	...., .. → ...., x ;
	...., (NOT x), .. → .... ;
[→ IMP]	...., x → ...., y, .. ;
	.... → ...., (x IMPLIES y), .. ;
[IMP →]	...., y, .. → .... ; | ...., .. → ...., x ;
	...., (x IMPLIES y), .. → .... ;

The first rule in the table is the *left* AND *elimination* rule, called [AND →] for short. It is used to eliminate the AND connective that appear on the left side of a sequent. It says in effect that any time an AND expression, (x AND y), is found on the left side of a sequent, separated by commas from other expressions, then this AND may be eliminated by using the transformed sequent shown on top, where the AND is replaced by commas keeping all the rest unchanged*. Clearly, this is consistent with our interpretation of sequents shown in [L1] above: The commas on the left side do stand for AND's. Thus, if the bottom sequent is valid then so is the top one, and if the bottom sequent is not valid then so is the top one. Therefore this rule does preserve validity. The dual of this rule is the *right OR elimination* rule, [→ OR]. Here OR is eliminated by commas.

The *right AND elimination* rule, [→ AND], may be interpreted as follows: If there is a problem in which one wants to prove (x AND y), then this problem may be split into two problems, one to prove x, and the other to prove y, keeping all the rest in the problem sequent unchanged. If these two smaller problems are both proven valid then clearly the original problem is valid. If at least one of the two smaller problems is not valid then the original problem is also not valid. Thus this rule also preserves validity.

The dual of this rule is the *left OR elimination* rule, [OR →]. Here also the problem sequent is split into two subproblems. This may be interpreted as follows: A conclusion from (x OR y) is valid if and only if the conclusion from x alone is valid, and the conclusion from y alone is also valid, keeping all the rest unchanged. Notice that (x OR y) can be true in three ways: When x is true and y is false, y is true and x is false, and both x and y are true. To prove the conclusion for (x OR y) one should thus be able to prove it in all the above three cases. All these three cases are covered by the two subproblems generated by the [OR →] rule.

The validity of the negation rules [→ N] and [N →] are not quite as obvious as the validity of the above rules. They say that a negated expression on one side of a sequent may just be moved to the other side after removing the negation, and this would preserve the validity of the transformation. To see why this is true let us consider [→ N] rule. Suppose the bottom sequent in [→ N] was valid. Then the conjunction of the expressions on the left side (let us call this C1) implies the disjunction of the expressions (let us call this (D1 OR (NOT x))) on the right in all world states, i.e.,

[2.1]: [C1 IMPLIES (D1 OR (NOT x))].

---

* In DL one may have expressions of the form (x1 AND x2 ... AND xn). In CK-LOG the [AND →] rule shown in table 2-1 is extended to cover this case. To simplify the presentation I have not shown the general case in this table.

Since the transformation preserves validity, it follows that

[2.2]: [(C1 AND x) IMPLIES D1]

should also be true in all the world states. It is not hard to show that [2.1] is valid IFF [2.2] is also valid.

The [→ IMP] rule may be derived by transforming the implication (x IMPLIES y) to ((NOT x) OR y) and applying the rules ([→ OR], [→ N]) in sequence to the resulting sequent. Similarly, the [IMP →] rule is obtained by applying ([OR →], [N →]) to the transformed implication. One may similarly construct also the [IFF →] and [→ IFF] rules. They are not shown in table 2-I.

All the rules shown in table 2-I preserve the validity of the sequents. Also note that each rule eliminates the occurrence of a logical connective from a sequent. After the application of any of these rules the resultant sequent(s) will be simpler in the sense that they will contain one less logical connective. Since any finite propositional problem will contain only a finite number of connectives in it, repeated applications of these rules to a problem should thus ultimately result in sequents that contain no logical connectives whatsoever. After this point is reached no more rules could be applied to any of the sequents. At this point one may test whether each of these terminal sequents is an axiom or not. If all of them are axioms then the initial problem is valid. If any of the terminal sequents is not an axiom then the initial problem is not valid. The simple examples shown in figure 2.2 illustrate this process. The rules applied to each problem sequent are shown in this figure to the right of the sequent in the order they are applied. The terminal sequents are both axioms in the first problem. Thus the root problem is valid. The terminal sequents 2a and 3 in the second problem are both not axioms. Here the root problem is not valid. Let me now present the rules for reducing quantified expressions.

2.4. Calculus of Sequents for Quantified Expressions.

The rules are shown in table 2-II. There are four of these. The first rule in this table is the *existential instantiation* rule, called [E →] for short. This replaces an existentially quantifier expression of the form, ((EXISTS x **range**) **exp**), by the two expressions '(elc ∈ **range**), (**subst** elc x **exp**)'. Here elc is a *new variable* not used previously in the deduction tree in which the bottom sequent of the rule appears as a

PROOF TREE 1:

4a. x, y → x •   4b. x, y → y •.

3. x, y → (x AND y) ; [→ AND]

2. → (x AND y), (NOT x), (NOT y) ; [→ N],[→ N]

1. (NOT (x AND y)) → ((NOT x) OR (NOT y)) ; [N →], [→ OR]

PROOF TREE 2:

3. x → ;

2a. → x ;   2b. → (NOT x) ; [→ N]

1. → (x AND (NOT x)) ; [→ AND]

Figure 2.2: Examples of Proofs in Propositional Logic.

problem. This variable will eventually be bound to a constant if and when the proof is completed, i.e. when all the terminal sequents become axioms. It is the creation of this constant that gives this rule its name, existential instantiation. This constant is required to be a member of the set specified by the **range**. Also, this constant is a function (is dependent) on all the variables uGx and uIc that might already appear in **exp**; these variables might have been introduced into **exp** by previous applications of the [→ U] and [U →] rules in the deduction tree. This dependency is indicated by the binding condition in the rule,

'eIc == (FEI uGx's & uIc's in **exp**)',

where 'FEI' is the name of a dummy function. It is used just to express the fact that eIc is dependent on its arguments. FEI is called the *Skolem function.* **Subst** is a function which is executed at the time of application of this rule. It substitutes eIc for every occurrence of the variable, x, in **exp**. The effect of this rule is to eliminate the 'EXISTS' quantifier from the sequent.

The *existential generalization* rule, [→ E], is quite similar to the existential instantiation rule. Here also the quantifier expression is replaced by two new expressions similar to the ones used above. But, in this case instead of substituting the quantifier variable by the new variable eIc, it is substituted by the new variable eGx. eGx is called the generalization variable. The value of this variable is also required to be in the set specified by the **range** given in the quantification, and is dependent on the variables uGx and uIc that already appear in the expression, as specified by the binding condition in the rule. Here the binding condition uses the *Skolem function* FEG.

The ∈-expressions appearing in the rules in table 2-II are given special interpretation during variable binding process. They are used to restrict the possible bindings for the variables. In cases where the **range** is a set of constants an attempt will be made to use the constants directly to bind variables in the expressions. If **range** is not a set of constants, then they can be either a union of concepts, X, or they may be expressions of the form '(r x)' where (r x) = {y | (r x y)}, or combinations of these.

TABLE 2-II: QUANTIFIER ELIMINATION RULES

Rule Name	Inference
[E →]	*Existential Instantiation*
	.., (eIc ∈ **range**), (**subst** eIc x **exp**), ... → .... ; Binding: eIc = (FEI uGx's & uIc's in **exp**).
	.., ((EXISTS x **range**) **exp**), ... → .... ;
[→ E]	*Existential Generalization*
	.... → ..., (eGx ∈ **range**), .. ; .... → ..., (**subst** eGx x **exp**), .. ; Binding: eGx = (FEG uIc's & uGx's in **exp**).
	.... → ..., ((EXISTS x **range**) **exp**), .. ;
[→ U]	*Universal Instantiation*
	...., (uIc ∈ **range**) → ..., (**subst** uIc x **exp**), .. ;
	...., → ..., ((EVERY x **range**) **exp**), .. ;
[→ U]	*Universal Generalization*
	..., (**subst** uGx x **exp**), .. → .... ; ..., .. → (uGx ∈ **range**), ... ;
	..., ((EVERY x **range**) **exp**), .. → .... ;

The difference between eIc and eGx is the following: Whereas to bind eIc one may create a new constant, ***eGx is required to be bound only to a constant that has been already created in the deduction tree***, i.e. a constant that was created in the deduction tree before the application of this rule. Similarly we also have the ***universal instantiation*** and ***universal generalization*** rules called [→ U] and [U →], respectively. The universal instantiation and generalization variables are independent variables, i.e. they are not functionally dependent on any of the variables that might already appear in **exp**. Notice that [U →] and [→ E] split the problem sequent into two new sequents. Also, it should be noted that each application of a quantifier elimination rule eliminates only the outer most quantifier in a quantified expression.

Repeated application of these rules will eliminate all the quantified expressions from the problem sequents and replace them by the appropriate propositional expressions. But the naming scheme used for the variables preserve the nature of quantification associated with the variables, and the *Skolem functions* keep track of dependencies among the various variables.

A general rule that governs the application of the rules given in tables 2-I and 2-II is that

> quantifier elimination rules may be applied to a problem sequent only after applying to the problem sequent all applicable propositional rules†.

Thus, after each application of a quantifier elimination rule one should apply to the resultant sequent(s) all the applicable propositional rules before applying the next quantifier elimination rule. The quantifier rules presented in table 2-II are different from the ones that are usually used in Gentzen's system [Kanger 1963][Bowen 1982]. Let me first explain the conventional rules and proof termination tests (axiom tests), and then contrast this with the new ones presented above together with the *mating algorithm* that is used to test proof terminations.

Normally, the quantified expressions themselves are not removed from a sequent. Everytime a quantifier rule is applied to a quantified expression, **qexp**, a copy of **qexp** is introduced into the sequent on the side of the sequent where **qexp** is present, the outermost quantifier in the copy is removed and all occurrences of the outermost quantified

† There are several exceptions to this rule. I will not enumerate them here. In the proof method illustrated later in this section I use one such exception, where all applicable quantifier rules are applied simultaneously to a sequent, instead of strictly following the above rule.

variable in the copy is replaced by a new variable (or a new constant, as the case may be). The level in the deduction tree at which this new variable or a constant was created is an important parameter that is kept associated with that constant or variable. The quantified expression itself is kept in the sequent for repeated future use if necessary. After the application of the quantifier rules all the applicable propositional rules are applied to the resultant sequents in the frontier set.

At this point an *axiom test* is given to the sequents in the frontier set. This test consists of substituting each generalization variable in the frontier set by an instantiated constant and examining the sequents in the frontier set after this substitution to see whether all of them are axioms. If they are not then another possible substitution is tried, and this process is continued until all possible substitutions are exhausted. If under one of these substitutions all the sequents in the frontier set become axioms, then this indicates the successful termination of the proof. If some of the sequents in the frontier set are not axioms, then the quantified expressions (if any) in the sequents are further expanded by reapplication of the quantifier rules, and the entire process presented above is iterated until, hopefully, a termination is achieved.

The assignment of constants to the variables is controlled by the following important restriction:

> A variable, uGx or eGx may be assigned a constant, eIc or uIc only if the level in the deduction tree at which the constant was created is not higher than the level at which the variable was created. If all the variables generated at the lowest level of the deduction tree are generalization variables then one or more of them may be used as instantiation variables.

As long as this assignment restriction is obeyed the proof process described above is complete and consistent (see [Kanger 1963] [Bowen 1982] for details). A significant defect in this algorithm is that the number of possible substitutions for variables can get astronomically large. The algorithm presented in [Bowen 1982] seeks to combine *unification* with the rule application process above to reduce the possible combinatorial explosion. The *mating algorithm* presented below provides another alternative to efficiently search for proof terminations. The total elimination of quantified expressions from the sequents makes it possible to use an efficient scheme to keep track of the variables, their substitutions and bindings.

In the variation introduced here constants are not created right away. Instead to represent the constants that might be created later, in the binding process discussed below, new instantiation variables are introduced. As we shall see below in the discussion of the *mating algorithm* the variables obtained after the elimination of all the quantifiers are kept in the sequent for repeated future use if necessary. Thus, a generalization variable uGx may be repeatedly regeneralized by creating new variables during the mating process, at different levels of the deduction tree. For a variable, eGx = (FEG ...), if (FEG ...) is undefined for the current bindings of its argument then eGx is also regeneralized. Similarly, a universal instantiation variable may be repeatedly reinstantiated, and for an eIc = (FEI ...), if FEI is undefined for the current bindings of its arguments, then eIc is also reinstantiated. The *Skolem functions* thus indicate when the existential variables should be regeneralized and when the existential constants should be reinstantiated. In the discussion below, the levels at which the regeneralizations and reinstantiations occur are kept track of by the subscripts associated with the variables and the constants. It is not hard to show that there is a proof in the conventional scheme iff there is a proof in the variant.

The *mating algorithm* used for axiom tests is best explained in the context of an example. This is done in the next section.

2.5. Mating Algorithm for Testing Axioms.

Let us consider the following problem**:

E1 = ((EVERY x)(NOT (f x x))),
E2 = ((EVERY x)(EXISTS y)(f x y)),
E3 = ((EVERY x)(EVERY y)(EVERY z)([(f x y) AND (f y z)] IMPLIES [g x z])).
E4 = ((EVERY x)(EXISTS y)(g x y)).

Problem, [P3]: E1, E2, E3 → E4 ;

Substituting the expressions in [P3] we get,

** One may understand what this problem says by substituting 'father-of' for 'f' and 'grandfather-of' for 'g' in the expressions.

[2.3]: {(EVERY x)(NOT (f x x))},
{(EVERY x)(EXISTS y)(f x y)},
{(EVERY x)(EVERY y)(EVERY z) ([(f x y) AND (f y z)] IMPLIES [g x z])}
→ {(EVERY x)(EXISTS y)(g x y)} ; [U →](5), [→ U]

This is the sequent at the root, which is level 1 of the deduction tree. There are no propositional rules that can be applied to this sequent. The quantifier elimination rule [U →] may be applied five times, as indicated in the sequent above, and [→ U] rule once. This will result in the following:

[2.4]: [NOT (f uGx11 uGx11)], [(EXISTS y)(f uGx21 y)],
[([f uGx31 uGy31] AND [f uGy31 uGz31]) IMPLIES (g uGx31 uGz31)]
→ [(EXISTS y)(g uIc41 y)] ; [N →],[E →],[IMP →],[→ E]

This generates the variables uGx11, uGx21, uGx31, uGy31, uGz31 and uIc41. All these variables are generated at level one. This is indicated in the naming of the variables by the last integer, which is 1. The first integer is used to indicate the expression in the problem sequent in which it was generated. Thus uGx11 was generated in the expression E1, uGx21 in E2, etc. At this point the rules indicated to the right of ';' in sequent [2.4] above are applied to this sequent resulting in the sequents below, and binding conditions which use the FEI and FEG functions:

[3a]: [f uGx21 eIc22],[g uGx31 uGz31] → [f uGx11 uGx11],[g uIc41 eGy4j] ;
Bindings: eIc22 = (FEI uGx21), eGy4j = (FEG uIc41).

[3b]: [f uGx21 eIc22] → [g uIc41 eGy4j],[f uGx11 uGx11],[f uGx31 uGy31] ;

[3c]: [f uGx21 eIc22] → [g uIc41 eGy4j],[f uGx11 uGx11],[f uGy31 uGz31] ;

Notice that the existential generalization* on the right side is represented by the variable eGy4j, with an unknown level j. This follows from the level condition associated with eGxij in table 2-III below, which specifies that $j \geq 2$, but does not fix a level for it. There are no more rules that one can apply to the sequents at level 3. It is time to do the axiom tests. Let me first state the rules that specify the conditions for assigning

* In general one will use function names FEG_i and FEI_j with increasing subscript numbers i and j in the binding conditions, if these functions are used several times in a deduction tree.

values to the generalization and instantiation variables, and the conditions for matching atomic expressions in a sequent. These conditions are shown in tables 2-III and 2-IV. The tables are self explanatory. It is useful to read these tables first before proceeding further with the text.

TABLE 2-III: INTERPRETATION OF VARIABLES.

Variable	Interpretation
uGxij	Universal generalization variable generated at level, j, and associated with the expression Ei in a problem sequent. This variable is generalized at any level, k $\geq$ j, by creating a new level k variable, xik, and making the assignment [uGxij $\leftarrow$ xik], while performing a match. The level of uGxij will then be equal to k.
eGxij	Existential generalization variable generated at level, j, and associated with the expression Ei in a problem sequent. Its value is always equal to the binding condition, (FEG uIc's & uGx's) specified at the time of generation of the variable, for the current bindings of the arguments uIc's and uGx's. The level j of eGxij is $\geq$ 1 + (maximum of levels of the arguments uIc's & uGx's).
uIcij	Universal instantiation variable generated at level, j, and associated with the expression Ei in a problem sequent. Is instantiated at any level k $\geq$ j, by creating a new constant, cik, and making the assignment [uIcij $\leftarrow$ cik], while performing a match. The level of uIcij will then be k, the level of cik.
eIcij	Existential instantiation variable generated at level, j, and associated with the expression Ei in a problem sequent. Its value always satisfies the binding condition, eIcij = (FEI uIc's & uGx's), specified at the time eIcij was generated. The level of eIcij is equal to 1 + (maximum of the levels of the uIc's & uGx's). At any level, k $\geq$ j, while performing a match, if (FEI uIc's & uGx's) is undefined and the level of eIcij is k, then eIcij is reinstantiated by creating a new constant cik and making the assignment [eIcij $\leftarrow$ cik].
(f ...)	This is a function term where f is a function name other than FEG and FEI. Its arguments may be constants, variables or other function terms.

To test for axioms one has to find a matching pair of atomic expressions in a problem sequent with one of the pair on the left side of the sequent and the other on the right side. The atomic expressions will always have the form of the relational expressions in the language DL (these are called *atoms*). The algorithm used for finding a match is called the *mating algorithm*, because as we shall see below, when a mating succeeds it will spawn new atoms in the sequents at the leaf nodes. The mating conditions are shown in table 2-IV.

TABLE 2-IV: CONDITIONS FOR MATING.

1. A level j **term**, t-ij, is either a level j variable or constant, or a level j function term, or a level j FEG or FEI expression. t-ij is *grounded* if it is a constant, or an FEI expression, or if it is a function term and all its arguments are grounded.

2. A level j variable, x-ij, can match a level k *grounded term*, t-hk, only if $j \geq k$. A successful match will generate the substitution [x-ij/t-hk], in which x-ij is substituted by t-hk.

3. A level j variable x-ij can match any level k variable y-hk. The match will generate the substitution [x-ij/y-hk] where $j \geq k$.

4. A variable, x-ij, can match any level k ungrounded function term t-hk only if none of the grounded arguments of t-hk has a level $>$ j. A successful match will produce the substitution [x-ij/t-hk].

5. Two function terms, (f ...) and (g ...) can match only if f $\equiv$ g, and the arguments match. It will produce the substitution corresponding to the substitutions of the arguments of the matched function terms.

6. If eGxij = (FEG uIc's & uGx's), then a match with (FEG ...) will succeed only if the match with eGxij succeeds.

7. If eIcij = (FEI uIc's & uGx's), then a match with (FEI ...) will succeed only if the match with eIcij succeeds.

8. Two **atoms** match only if they have identical relation names (tuple names) and their arguments match.

9. At no point may a level j variable, x-ij, become equal to a level k grounded term for $k > j$, due to matching processes alone. (They could be equal if the sequent itself had in it an equality expression declaring them to be equal. The inference rule for equality is not discussed here.)

Let us now consider the possible matches in the leaf sequents of our example to see how the above rules are applied. The sequent [3a] is reproduced below:

[3a]: [f uGx21 eIc22],[g uGx31 uGz31] → [f uGx11 uGx11],[g uIc41 eGy4j] ;
Binding: eIc22 = (FEI uGx21).

If [f uGx21 eIc22] is mated with [f uGx11 uGx11] then uGx21 will become equal to eIc22, which is forbidden by the rule 9 in table 2-IV. So the only possible match in [3a] is between [g uGx31 uGz31] and [g uIc41 eGy4j]. When this match is done the inference engine will create a new constant, say c41, make the following assignments and note the binding and substitutions† below:

Binding: [eGy4j = (FEG uIc41)].
Assignment: [uIc41 ← c41].
Substitutions: [uGx31/c41, uGz31/(FEG c41)]

When the above substitution is now performed on the mated pair, each member of the mated pair will yield through the substitution the same new atom, (g c41 (FEG c41)). This is called the *offspring*. This offspring is now added to the sequent on its left and right sides causing the sequent to become an axiom. The *parents* of this offspring are still kept in the sequent. The same substitution is performed on all the other sequents in the frontier set. This may result in the generation of several offsprings. Each sequent in the frontier set is updated with the offsprings generated in the sequent through the substitution. Thus a given successful mating may result in the generation of several offsprings. The resulting modified sequents at level 3 are shown below:

[3a-1]: [f uGx21 eIc22],[g uGx31 uGz31],[g c41 (FEG c41)]
→ [f uGx11 uGx11],[g uIc41 eGy4j],[g c41 (FEG c41)] •

[3b-2]: [f uGx21 eIc22] →
[g uIc41 eGy4j],[g c41 (FEG c41)],
[f uGx11 uGx11],[f uGx31 uGy31],[f c41 uGy31] ;
Bindings: eIc22 = (FEI uGx21), eGy4j = (FEG uIc41).

[3c-2]: [f uGx21 eIc22] →
[g uIc41 eGy4j],[g c41 (FEG c41)],
[f uGx11 uGx11],[f uGy31 uGz31],[f uGy31 (FEG c41)] ;
Bindings: eIc22 = (FEI uGx21), eGy4j = (FEG uIc41).

In sequent [3b-2] the inference engine will now first search the left side for a match with the newly introduced offspring on the right side and thus choose to match [f c41 uGy31] with [f uGx21 eIc22]. Here uGy31 (a level 1 variable) should match with eIc22 (a level 2 constant). But as per rule 2 in table 2-IV this is possible only if uGy31 is regeneralized.

† In constructing substitutions always variables are substituted by constants, or a variable at level, i, is substituted by another at a level less than or equal to i, or a variable is substituted by an (FEG ...) expression, or an FEG expression is substituted by an FEI expression, or a variable is substituted by a term of the form '(F ...)' where F is a function name. Every function, F, used in CK-LOG should be declared to MDS. The interpretation given by MDS to F is defined as a lambda expression in LISP.

Thus the system will create a new variable, say y32 and assign it to uGy31 and then perform the mating. This will result in the following bindings, assignment and substitutions:

Binding: [eIc22 = (FEI uGx21)].
Assignment: [uGy31 ← y32],
Substitutions: [uGx21/c41, y32/(FEI c41)].
Binding: [(FEI c41) = c22].

The system now creates a new constant c22 because, as per conditions given above for assigning values, eIc22 = (FEI c41), and (FEI c41) is at this point undefined and thus a new constant is called for. The incorporation of these substitutions in the leaf sequents will now result in the following augmented sequents:

[3b-4]: [f uGx21 eIc22], [f c41 c22)] →
[g uIc41 eGy4j], [g c41 (FEG c41)], [f uGx-11 uGx-11],
[f uGx-31 uGy-31], [f c41 uGy31], [f c41 c22)] •

[3c-5]: [f uGx21 eIc22], [f c41 c22] →
[g uIc41 eGy4j], [g c41 (FEG c41)], [f uGx11 uGx11],
[f uGy31 uGz31], [f c22 (FEG c41)] ;

Here [3b-4] is an axiom. In [3c-5] a match will now be attempted for the newly introduced offspring [f c22 (FEG c41)]. This will cause the system to mate [f c22 (FEG c41)] with [f uGx21 eIc22]. The mating process here will first generalize uGx21 to x22, in order to match it with c22, and obtain the substitution [x22/c22]. Then eIc22 will be equal to (FEI c22) which is at this point undefined. Thus a new constant c23 will be created. Notice that in this mating, (FEG c41) can match with (FEI c22) because the level of (FEG c41) can be any number that is greater than 1 as per rule given in table 2-III for variables eGxij. As a result of the match here (FEG c41) will acquire the level 3. The results are shown below:

Binding: [eIc22 = (FEI uGx21)]
Assignment: [uGx21 ← x22].
Substitutions: [x22/c22, (FEG c41)/(FEI c22))]
Binding: c23 = (FEI c22) = (FEG c41).

[3c-6]: {[f uGx21 eIc22],
[f c41 c22], [f c22 c23] →
[g uIc41 eGy4j], [g c41 (FEG c41)],
[f uGx11 uGx11], [f uGy31 uGz31],
[f c22 (FEG c41)],
[f c22 c23] •

These substitutions now reduce all the leaf sequents to axioms and the proof is done. Let us now take a look at what has happened by just listing the matched atoms in the various leaf sequents:

[3a-1]: [g c41 c23] → [g c41 c23)] •
[3b-4]: [f c41 c22] → [f c41 c22] •
[3c-6]: [f c22 c23] → [f c22 c23] •

If 'f' is the 'father-of' relation and 'g' is the 'grandfather-of' relation then the above result may be paraphrased as follows:

> c22 is the father-of c41, c23 is the father of c22, and c23 is also the grandfather of c41, for any c41.

In general, the search for mates in each leaf sequent is very much like the search for *unifying literals* in *resolution theory* [Robinson 1965]. But unlike resolution theory the search is local to the sequent. Also, instead of searching for the *most general substitution* one searches here for the *most specific substitution*. Whereas in resolution theory when two clauses are unified the matched literals are deleted, here when a mating succeeds the sequents in the frontier set may get new offsprings added to them. Thus, the mating algorithm illustrated above is the dual of the unification algorithm. The mating process used for axiom tests is a bit more complicated than the unification algorithm, because of the need to make assignments and test bindings and level conditions. The presence of floating levels, as in the variable eGy4j, can at tîmes complicate the level checking problem. But in most cases this problem will be simple. The benefit that one gains for this extra work is that it may reduce the search space of possible matings.

In the example above the candidate pairs for the matings in each sequent was unique. In general, of course, this will not be the case. There may be several choices available for mates, leading to a possible search explosion, and need for *backtracking* when a chosen set of choices fail. Failure will occur in an axiom test for a sequent when level checking fails for all the mated pairs in the sequent, or when there are no candidates available in the sequent for mating, or as we shall see in the next subsection, a contradiction is detected in the world state associated with the sequent.

This basic scheme of proof construction is further modified in CK-LOG's inference engine to facilitate communication with the knowledge base and the world states. These modifications are discussed in the next subsection. The mating algorithm illustrated above for axiom testing and variable substitution, and the modifications presented in the next subsection, are unique to this inference engine and are new. They define a variant of the method first proposed by Kanger [Kanger 1963]. The proof procedure is *complete* (in the sense that it can find the proof for every valid sequent) and *consistent* (in the sense that every time the proof terminates successfuly the root problem is valid). Let me now briefly outline how the above procedure has been modified to facilitate interaction with the world state and the knowledge base.

2.6. Communication Between Inference Engine, Knowledge Base and World States.

The communication between the inference engine, the knowledge base and the world states is intended to serve two basic purposes:

1. To update problem sequents at the leaf nodes of a deduction tree with knowledge from the knowledge base **K[U]**,
2. To update the world state based on the findings resulting from the axiom tests performed on the leaf sequents in the deduction tree.

I will present the scheme first and then illustrate it using the example discussed above. Let me here assume that the problem sequents do not contain any operator expressions. Thus, to create new constants or to make a relation true (false) in the world state, I am assuming that no actions would be invoked. To introduce them into the world state they will be simply asserted into the world state.

Each problem sequent, Q, in the deduction tree will have a unique world state, $\mathbf{U}_t$, associated with it. It will also have associated with it a set of problems called the *hypothesis problems*, denoted by H[Q]. The hypothesis problems associated with Q will depend on information that is unknown in the world state, but is needed to solve Q. They are generated during the problem solving process as discussed below. The *problem state* defined by Q will consist of the following:

[Q, (predecessor-of Q), (successor-of Q), (binding-conditions-of Q)
(assignments-of Q), (substitutions-of Q), (world-state-of Q),
(hypothesis-problems-of Q)].

The *context* of a world state, U_t, is the set of sequents defined by,

$$(\text{context-of } U_t) = \{Q \mid (\text{world-state-of } Q\ U_t)\}.$$

Let F be the set of sequents in the leaf nodes of the deduction tree. F is the *frontier set* of the deduction tree. At the beginning of the problem solving process the frontier set, $F_0 = \{Q_0\}$, where Q_0 is the problem sequent at the root of the deduction tree. The world state, U_0, associated with Q_0 will be the *current world state* that existed at the time this problem was posed.

When the successors of Q_0 are spawned the frontier set of the deduction tree will change. Let $\{F_1\}$ be this new frontier set. The context of the world state, U_0, will then change to $[\{Q_0\} \cup F_1]$. This process of changing the context of the world state will continue as the deduction tree grows. So far the world state itself has not changed. Changes to the world state may occur when an axiom test is performed on one of the leaf sequents in F.

Before performing the axiom test the sequents Q in F will at first be augmented with information obtained from the knowledge base K[U]. The nature of this augmentation will depend on the logical restrictions (consistency conditions) in K[U], that are associated with the atoms that appear in Q. The augmentation process is described below.

Let A = (r x y) be an atom that appears on the right side of a problem sequent Q in {F}, separated by commas from the rest of the expressions on the right side as in

[Q]: ... → .., (r x y), ... ;

Let x be an instance of the concept X, y an instance of Y. Then (**r** X Y) is called the *dimension* of (**r** x y)**. This dimension may have a *consistency condition* (a logical restriction) associated with it in the knowledge base **K[U]**. This consistency condition may in general have one of three forms:

[2.5]. [(**r** x y) IFF **exp**],
[2.6]. [(**r** x y) IMPLIES **exp**], or
[2.7]. [**exp** IMPLIES (**r** x y)]

where **exp** is a *logical expression without modal expressions*, in which x and y appear as the only *free variables*. The variables are interpreted as being universally quantified. **Exp** will state the logical restrictions on the atom (r x y). The way Q gets augmented will depend on whether the consistency condition associated with the dimension (**r** X Y) is of the type [2.5], [2.6] or [2.7] above. For an atom that appears on the right side of a problem sequent the three cases are shown in table 2-V by the first three inference rules. The last three show the rules for an atom on the left side. Cases [2.5] and [2.7] have identical effects: Q is simply replaced in the frontier set, **F**, by the sequent shown on top of the inference rules. In case [2.6], Q is replaced by the two sequents on top. The inference engine will keep track of the augmentation done in this manner to make sure that no sequent gets augmented twice for the same atom. I will refer to this process of retrieving conditions from **K[U]** as the **KB**-lookup process. The **KB**-lookup process is used to interpret all the restrictions associated with the dimensions in **K[U]**.

I have not discussed above all the tasks performed in the **KB**-lookup process. The retrieval of all the information pertinent to a given (r x y) may involve more work than what has been described above. In general, an atom may contain terms, (f ...), or terms like '(father-of x)', '(grandfather-of x)', etc., instead of containing only variables. These terms should be given proper interpretations during the **KB**-lookup process. Also dimensions (r X Y) may have a restriction that specifies an upper bound and a lower bound on the number of distinct instances, y of Y, to which a given instance, x of X, may be related to via **r**. Thus, for example, by placing a (1, 1) after the dimension one may indicate the uniqueness of the relationship between x and y. I will not present here

** The ideas presented here on dimensions and consistency conditions are discussed in greater detail in sections 3.3, 3.4 and 4.1.

TABLE 2-V: RULES FOR PROBLEM AUGMENTATION.

CC-type	Inference Rule for Augmentation.
[2.5]	... → .., (**r x y**), **exp**, ... ;
	... → .., (**r x y**), ... ;
[2.6]	..., **exp** → .., (**r x y**), ... ; ... → .., (**r x y**), ... ;
	... → .., (**r x y**), ... ;
[2.7]	... → .., (**r x y**), **exp**, ... ;
	... → .., (**r x y**), ... ;
[2.5]	.., (**r x y**), **exp**, ... → .... ;
	.., (**r x y**), ... → .... ;
[2.6]	.., (**r x y**), **exp**, ... → .... ;
	.., (**r x y**), ... → .... ;
[2.7]	.., (**r x y**), ... → ....; .., ... → **exp**, ... ;
	.., (**r x y**), ... → .... ;

the full details on these. Some of them are illustrated later in this subsection.

After modifying each Q in the frontier set, **F**, in this manner the axiom test will now be performed on each Q. Let Q1 be the sequent in **F** on which the axiom test was first done. If this test resulted in reducing Q1 to an axiom, then Q1 will have a matching pair of identical offspring on either side of →. Also, this test might result in the introduction of other offsprings in other sequents that exist in **F**. Let $\mathbf{F}_S$ be the subset of **F** which were thus changed.

If during the axiom test new constants were created then each such new constant will be instantiated in the world state, taking care that it satisfies the appropriate **range** restriction specified in the sequent. If all the arguments of an offspring, A, resulting from the mating in Q1, are constants (such an atom is called a *grounded atom*), and the same offspring appears on both sides of a sequent in $\mathbf{F}_S$, then A will be asserted into the world state.

These assertions and the creation of constants will result in the creation of a new world state. The context of this new world state will be $\mathbf{F}_S$, the subset of sequents in **F** which were changed by the axiom test. If all the assertions and the creation of constants are unconditionally accepted in the new world state then the axiom test performed on

Q1 is considered to be successful and the proof process is continued.

If they are conditionally accepted, then the modeling system will return a logical expression, say **C**, which expresses the condition under which the modifications performed on the world state will be acceptable. **C** will represent the information in the world state that is relevant to the problem being solved, but is currently unknown in the world state. The system will now assume **C** as a hypothesis, by introducing the problem,

$$[Q_{h1}]: \rightarrow (\text{ACHIEVE C})\ ;$$

as a new problem into the set **H**[Q1], the set of hypothesis problems associated with Q1. The world state of Q_{h1} will be the new world state associated with Q1 after the modifications. Also, Q_{h1} will have the same bindings, assignments and substitutions associated with it as Q1. This hypothesis problem will also be added to the frontier set $\mathbf{F}_S$.

If the new world state has a contradiction in it then the axiom test is considered to have failed. In this case either another possible mating in Q1 will be attempted, or the problem solving system will backtrack to a previous problem state and continue the deduction.

The problem solving process will terminate when it encounters a frontier set of axioms. In this case the problem is said to be *unconditionally solved.* At this point each axiom, Q, in the frontier set will have a world state associated with it. Each distinct world represents a solution to the root problem. If one or more of the hypothesis problems remain unsolved, then the problem is said to be *conditionally solved*: under the condition that the hypotheses are true. User will then have the choice either to continue the problem solving process to solve the hypothesis problems or terminate the process at that point. If any of the problems lead to a contradiction in the world state for all possible choices of matings then the problem is *not solved.*

This modified mating algorithm achieves four purposes:

1. It uses the knowledge base to augment the problems with information pertinent to the solution of the problems.

2. It uses the offsprings produced by the matings to update the world state and check the consistency of the matings themselves.
3. It identifies the set of hypotheses under which the solution holds true.
4. It produces the world states associated with the frontier set of axioms. these world states will represent the solutions to the problem.

I have here introduced an additional condition for a mating to succeed, the offsprings produced by a mating should not cause a contradiction in the world state. This additional condition will certainly preserve the *consistency* of the proof process described above (i.e. if a proof terminates successfuly then the root problem is valid). But to assure *completeness* (i.e. for the proof process to terminate successfuly for every valid sequent), clearly the specification of K[U] should be complete in a well defined sense. The concept of completeness of K[U] is captured by the condition of *locality* described below and further elaborated later in section 5.

The locality condition is expressed using the concepts of *relation paths* and *contradictions*: A list of relation names, $(r_1\ r_2\ \ldots\ r_n)$, is said to be a *relation path* in a world state iff there exist constants $c_1, c_2, \ldots, c_{(n+1)}$ such that

$$[(r_1\ c_1\ c_2)\ \text{AND}\ (r_2\ c_2\ c_3)\ \text{AND}\ \ldots\ \text{AND}\ (r_n\ c_n\ c_{(n+1)})]$$

is true (or unknown) in the world state. In this case we shall say that

$$(c_1\ (r_1\ r_2\ \ldots\ r_n)\ c_{(n+1)})$$

is true (or unknown) in that world state. The introduction of a new literal L_1 into a world state,

$$L_1 = \text{Either}\ (r\ c_1\ d_1)\ \text{or}\ (\text{NOT}\ (r\ c_1\ d_1)),$$

where c_1 and d_1 are constants, will cause a contradiction in the world state if the following is true in the new world state obtained after the introduction of the literal:

$$[(L_1\ \text{IMPLIES}\ L_2)\ \text{AND}\ (\text{NOT}\ L_2)],$$

where L_2 is another literal in the same world state,

$$L_2 = \text{Either } (r' \ c_2 \ d_2) \text{ or } (\text{NOT } (r' \ c_2 \ d_2)).$$

In this case we shall say that L_1 contradicts L_2 in the world state. The *locality* condition may now be expressed as follows:

> *Locality Condition*: If L_1 contradicts L_2 in a world state U_t, then there is a relation path $(r_1 \ r_2 \ ... \ r_n)$ such that $(d_1 \ (r_1 \ r_2 \ ... \ r_n) \ c_2)$ is true in U_t.

The satisfaction of this condition by K[U] is relevant to the retrieval of *all the logical conditions* that are pertinent to a given set of modifications performed on the world state. The condition in effect says that if two literals can potentially contradict each other in a world state, then they should be related to each other in some way (either directly or via a relation path) in K[U]. Thus K[U] is required to capture all potential relationships that exist in the universe. I will comment on this further in section 5.

An important property of the inference rules displayed here and later in section 2.9 is that the rules can be run both ways, from bottom to the top as well as from the top to the bottom, i.e. starting from a set of axioms one may apply the rules in the reverse order to conjecture the theorems that give rise to them. In fact this is the way Gentzen viewed the system he proposed. The significance of this is that one may use CK-LOG also as a *theory forming system* to form theories of situations that exist in a world state. In principle, this feature gives CK-LOG a capability to learn general principles from given specific situations. This problem needs further study.

Let me now illustrate the process described above by considering again the father-grandfather problem. The universe of this example is described by the concept, PERSON, shown below:

> *Structure*: (father-of PERSON PERSON), 1, *Irreflexive*
> (grandfather-of PERSON PERSON), 1

This says that every person has exactly one father (this is stronger than the condition specified by E2) and the father relationship is irreflexive. This captures the meaning of both expressions E1 and E2 presented in the last subsection. The condition of the expression E3 may now be stated as a restriction on the grandfather-of relation, as follows:

[(father-of (father-of person) x) IMPLIES (grandfather-of person x)].

This is a type [2.7] restriction. The problem to prove is,

→ ((EVERY x PERSON)(EXISTS y PERSON)(grandfather-of x y)) ;

The sequence of deductions on this problem is shown in table 2-VI below. In each row of this table the last column indicates the rule(s) applied to the sequent in the row, to get the sequent(s) in the next row below.

TABLE 2-VI: THE FATHER-GRANDFATHER PROBLEM.

Problems	Sequents	Rules Used.
[P1]	→ ((EVERY x PERSON) (EXISTS y PERSON)(grandfather-of x y)) ;	[→ U]
[P2]	(uIc11 ∈ PERSON) → ((EXISTS y PERSON)(grandfather-of uIc11 y)) ;	[→ E]
[P3]	(uIc11 ∈ PERSON) → (eGx1j ∈ PERSON) ; (uIc11 ∈ PERSON) → (grandfather-of uIc11 eGx1j) ; Binding: eGx1j = (FEG uIc11).	[**KB**-lookup] grandfather-of
[P4]	(uIc11 ∈ PERSON) → (eGx1j ∈ PERSON) ; (uIc11 ∈ PERSON) → (grandfather-of uIc11 eGx1j), (father-of (father-of uIc11) eGx1j) ;	[**KB**-lookup] father-of
[P5]	(uIc11 ∈ PERSON) → (eGx1j ∈ PERSON) ; (uIc11 ∈ PERSON), (father-of c11 c22) → (grandfather-of uIc11 eGx1j), (father-of (father-of uIc11) eGx1j), (father-of c22 eGx1j) ;	[**KB**-lookup] father-of
[P6]	(uIc11 ∈ PERSON) → (eGx1j ∈ PERSON) ; (uIc11 ∈ PERSON),(father-of c11 c22),(father-of c22 c33) → (eGx1j ∈ PERSON), (grandfather-of uIc11 eGx1j), (father-of (father-of uIc11) eGx1j), (father-of c22 eGx1j) ; Assignment: (uIc11 ← c11).	[Axiom-test]
[P7]	(c33 ∈ PERSON), (uIc11 ∈ PERSON) → (eGx1j ∈ PERSON), (c33 ∈ PERSON) • (uIc11 ∈ PERSON),(father-of c11 c22),(father-of c22 c33) → (eGx1j ∈ PERSON), (grandfather-of uIc11 eGx1j), (father-of (father-of uIc11) eGx1j), (father-of c22 eGx1j), (father-of c22 c33), (grandfather-of c11 c33) • Assignment: [eGx1j ← x13], [x13 ← c33] Substitution: [eGx1j/x13].	•

Sequent [P2] results directly from the application of [→ U] to [P1], and [P3] results from the application of [→ E] to [P2]. At this point **KB**-lookup is done for the grandfather-of relation resulting in [P4] as per the inference rule for type [2.7] conditions shown in table 2-V. Notice that when the condition associated with the grandfather-of relation is retrieved from the knowledge base the system makes the appropriate variable substitutions in the expression. At this point a second **KB**-lookup is done, this time for the father-of relation. The term '(father-of ulc11)' appearing in this relation will cause the system to look for the father-of the constant ulc11 in the world state. But ulc11 has not been yet created. Since ulc11 is a universal instantiation variable, this will cause the system to create a new instance of PERSON, say c11, assign it to ulc11 and record the assignment. When the system now looks for (father-of c11) in the world state it will notice the restriction, '1', placed on the dimension '(father-of PERSON PERSON)'. This is interpreted as saying that every PERSON has a unique father. This will cause the system to create another person, say c22, and assign c22 as the father of c11. It could not assign c11 as the father of c11 because the father-of relation has been declared to be *irreflexive.* This will result in the sequent [P5]*. At this point another **KB**-lookup is done, this time for the '(father-of c22 eGx1j)'. This will result in the sequent [P6]. The performance of the axiom test on this sequent will conclude the proof with [P7] in the table.

2.7. Assertions to World State Models.

Constants are created in a world state by the assertion,

(ASSERT (*instance-of* **range** c)) → ; or
(CRI c **range**) → ;

where **range** specifies the range restriction on the constant c. CRI (CReate Instance) is one of the TMS commands. By convention this command may appear only on the left side of a sequent (ASSERT may appear on either side). A CRI expression appearing on

* In general, a literal '(r (r1 x) y)' is interpreted as '((EXISTS z Z)([r1 x z] AND [r z y])),' where Z is the range of z. The standard way to analyze a literal with embedded terms like this is to replace the literal with its equivalent logical expression in the sequent, and then proceed with the analysis of the modified sequent. There are, however, a few special cases where, the series of inference steps implied by this general method, may be cut short. The case illustrated above is one of these special cases. Here the '(father-of PERSON PERSON)' relation had no general logical restrictions associated with it other than those specified by the flags '1, *irreflexive*'. The processing of the interpretations given to flags like these give rise to the various special cases. It is not hard to verify that the short cut presented above will in every case produce the same result as the general process for the universal instantiation variable ulc11, and for the restrictions specified by the flags '1, *irreflexive*'.

the right side has no interpretation. The offsprings generated in an axiom test are asserted by using the ASSERT command,

$$(\text{ASSERT } (\text{offspring}_1 , \text{offspring}_2 , ..., \text{offspring}_n)) \rightarrow ;$$

The inference engine will pass these commands directly for assimilation by the truth maintenance system. ASSERT statements on the left side of a sequent, like the one above or more general ones discussed later in section 2.9, are the analogs of Levesque's TELL operator. An ASSERT statement on the right side has a different interpretation. It is interpreted as an attempt to conclude a proven (or a hypothesized) assertion as explained below.

$$... \rightarrow (\text{ASSERT } (\text{offspring}_1 , \text{offspring}_2 , ..., \text{offspring}_n)) ;$$

> will be asserted into a world state only if for each atom in the ASSERT a matching atom can be found on the *left side* of the problem sequent, separated by commas from other expressions in the sequent.

In general, the argument of an ASSERT expression may contain both positive and negated atoms (literals).

> For a negated atom appearing as an argument of an ASSERT expression on the *right side* of a sequent, the negated atom will be introduced into the world state only if a matching atom without the negation also appears on the *right side*, separated by commas from all the other expressions on the right side.

If matching atoms do not exist in a sequent for the arguments of an ASSERT expression, then CK-LOG will present to the user the list of unmatched literals. If the user forces the assertion, then CK-LOG will enter the conjunction of the forced assertions as a hypothesis problem associated with the sequent. If the assertion contained unbound generalization variables then the system will attempt to bind them using the constants that exists in the world state associated with the sequent. If no such constants are available in the world state then user will be prompted to confirm whether new constants ought to be created to satisfy the asserted conditions. Universal generalization variables are interpreted as indicating that the assertion should hold for all the bindings in the world state over which the variables range, and existential generalization variables are interpreted as indicating that the assertion should hold for one or more bindings in the world state over the range of the variables. If unbound instantiation variables occur in an assertion then there would be in the assertion also commands to create new bindings for them. Otherwise the assertion cannot be executed. Assertions are made into a world state only after an attempt has been made to bind the variables that occur in the assertion. Some examples of assertions are discussed in section 4 and in [Srinivasan 1984].

If the argument of an ASSERT expression is itself a complex logical expression, instead of literal, then CK-LOG will use its inference engine to break the logical expression down to its components and find the appropriate assertions that it should make in the world state. The inference rules for this are stated in section 2.9.

2.8. The Action Calculus of CK-LOG.

The use of CRI and ASSERT will occur in the communication between TPS and TMS only if there are no actions involved in creating a constant in the world state or in making an atom true in the world state, or a user had suppressed the invocation of actions. If actions are involved then the CREATE expression is used to introduce offsprings into the world state, or to create new constants in the world state. CREATE expressions are the standard ones used by TPS to make assertions into world states. They are presented to the inference engine as a problems to be solved:

(CREATE (c ∈ **range**)) → ; or
(CREATE <conjunction of offsprings>) → ;

The inference rules used by the inference engine for interpreting modal expressions like the CREATE expression above are discussed in the next subsection. These rules ultimately reduce the modal expressions to a set of actions that are needed to make the expressions true, and ASSERT these actions into the world state. A new action will get created in a world state by the instantiation of a variable, elc or ulc whose range is an ACTION. Suppose x was an instantiation variable elc or ulc, and action is the new instance created and bound to x. Then TMS will update every sequent, Q, in the frontier set in which the variable x appears, with the following *action predicates*, [AP], by adding these predicates to the left side of Q:

[AP]: [(before (*time-of* (status-of action successful)))
 (**value-of** (*function* action))],
 [(before (*time-of* (**value-of** (*function* action)))
 (**value-of** (*behavior* action))],

where **value-of** is a function that returns the value of the term in the current world state*. The solution of these problems will assure that the intended creation of the constants and asserted literals is accomplished. If several actions are created by a parallel assertion, then the solution is considered successful only if the action predicates for all the actions succeed. This is indicated by associating a special flag (I will use flags $1, $2, etc. as flags to indicate this parallel success requirement) with the predicates that have such a special requirements. I will refer to this flag as the *parallel predicate flag*. Thus, in a problem,

[AP1], [AP2], ... → .... ;

it is quite possible that the problem terminates successfully with the realization of [AP1] alone (we will encounter such an example in section 4). If the same special flag is not associated with both [AP1] and [AP2] then normally one could terminate the problem at this point. But, if both [AP1] and [AP2] have a common *parallel predicate flag* then the problem solution will be continued until [AP2] is also successful.

In general, the analysis of a modal expression by the inference engine may result in the creation of actions that are needed to make the expression true in a future world state. The actions so created will result in augmenting the sequents in the frontier set of a deduction tree with action predicates with or without parallel predicate flags. CK-LOG's *action calculus* is defined by the proofs generated using these action predicates. An example of this calculus is discussed in section 4.

---

* It should be noted that this is not the only possible way to update the sequents in the frontier set. Different formulations are possible for the action predicate. It is not clear to me yet whether there is a general formulation for the action predicate that is most advantageous for the representation of actions and reasoning about them. If one considers world states with only one action at a time then the kind of action predicate that one chooses does not make much of a difference. If more than one action could exist in a world state at a time, then ones ability to simultaneously monitor multiple actions will be determined by the kinds of action predicates that one uses. More experience with different kinds of actions is necessary before the nature of this problem is better understood. The predicate shown here is adequate for our purposes here.

In these action predicates the *function* of an action plays the role of *post conditions* that are usually associated with actions in *situation calculus* [McCarthy 1969]. The preconditions for actions may occur in its *behavior* statement, in the logical conditions associated with the creation actions, and in the logical conditions (consistency conditions) associated with the relations that are used to describe the actions. The use of the literal '(status-of action successful)' in [AP] above gives one opportunities to define logical restrictions specific to an action instance. These will specify the conditions for the successful termination of an ACTION, in addition to the general conditions specified by the *function* and *behavior* of ACTION.

The inference rules used for analyzing modal expressions, that use operators like ACHIEVE, CREATE, DESTROY, etc., are discussed in the next subsection.

2.9. The Specialized Inference Rules.

I will present here the inference rules for a representative sample of operators used in TML*. The rules are presented below following the convention established in tables 2-I and 2-II. Unlike the rules presented in table 2-I and 2-II, each inference rule presented here has a condition associated with it. This condition is called the *invocation condition.* A rule may be invoked in a theorem proving process only if the invocation condition associated with it is true in the world state at the time of its invocation.

Bold items in the rules refer to *meta-syntactic* functions or patterns. Thus for example in the first rule, namely the *achieve-elimination* rule, '**value-of**' refers to the meta-syntactic function which returns the value of its argument in the world state at the time the rule is applied. If the value is undefined in the current world state then it will return the **value-of** expression itself as its value. '**Term**' is a syntactic pattern that stands for a function term, **sexp** denotes a simple logical expression with or without operators but with no time parameters, and (**tm-term exp**) denotes a timed expression where **tm-term** is the term that specifies the time. The pattern, **exp**, is used to denote an expression which may or may not be a timed one. Finally, **atm-sexp** denotes a untimed literal, and **atm-exp** denotes a timed or untimed literal. Each rule is followed

* In the inference rules presented in this section I have not indicated the presence of *Skolem functions* that specify dependencies between existential and universal variables. Throughout this section it is to be understood that appropriate *Skolem functions* are generated at the time of rule application, wherever necessary.

by a brief comment.

In all the inference rules the top statement may be viewed as defining the meaning of the bottom one. The way these rules are used in the action calculus of CK-LOG is illustrated by the example in section 4. Let me begin with the achieve elimination rule.

TABLE M-I: Achieve Elimination Rule.

Name	Condition	Inference
[ACH-EL]	T	(CREATE (**value-of term**))
		(ACHIEVE **term**)
[ACH-RED]	T	(ACHIEVE **term**)
		(ACHIEVE (ACHIEVE **term**))

This is called [ACH-EL] for short. The condition, T, indicates that this rule is unconditionally applicable. The absence of '→' symbol in the rule indicates that the same rule applies both to the left and the right side of a sequent: The rule may be applied uniformly without regard to where the ACHIEVE statement occurs in a sequent, even if it occurs embedded in another larger expression. In this sense, the rule is said to be context independent. Informally, one may think of the above rule as saying that for any **term**, (ACHIEVE **term**) always means, CREATE whatever the **value-of** the **term** denotes in the current world state. Most of the rules presented below are context independent rules. The achieve reduction rule, [ACH-RED], reduces embedded ACHIEVE expressions. The rules in the next table are used to eliminate ISTRUE, ISFALSE and ISUNKNOWN operators. Their interpretation is quite straight forward.

The ISTRUE, ISFALSE and ISUNKNOWN operators are used to query the world state models. For the ISTRUE and ISFALSE operators, if the requested information is unknown for a given atom then TMS will attempt to find a *default* value for it, if any (as discussed in section 3). It will return F only if the *default* value also does not exist. Notice that the rule [IS?-EL] is applied only if (ISUNKNOWN **term**) is true. If it is true then it is replaced by T, else it is left unchanged. There is no facility in TPS to reason with unknown expressions. Thus the UNKNOWN operator will not normally appear in problem statements. One could use this to introduce *default* assumptions in a problem. It may be noted that if an expression contains variables which have no bindings as yet specified, then the value of the expression in the world state will be ? (unknown), and in

TABLE M-II: ISTRUE, ISFALSE and ISUNKNOWN Elimination.

Name	Condition	Inference
[IST-EL1]	T	(**value-of term**)
		(ISTRUE **term**)
[ISF-EL]	T	[NOT (**value-of term**)]
		(ISFALSE **term**)
[IS?-EL]	(ISUNKNOWN (**value-of term**))	T
		(ISUNKNOWN **term**)
[IST-CR]	T	(ISTRUE **exp**)
		(ISTRUE (CREATE **exp**))
[IST-DES]	T	(ISTRUE (NOT **exp**))
		(ISTRUE (DESTROY **exp**))
And similar rules for [ISF-CR], [ISF-DES], etc.		

this case the ISTRUE and ISFALSE operators for that term will return F, and ISUNKNOWN will return T.

The rules in the next table give us a way of eliminating (*time-of* **event**) or (*interval-of* **event**) expressions from a sequent. These expressions are called *event-time* expressions. There are two cases corresponding to whether the **event** is true in the current world state or not. The meta-syntactic variable, **event-rel**, stands for '*time-of*' or '*interval-of*', and the variable **time-rel**, stands for 'after', 'before', 'during', 'between' or 'at.' It may be noted that 'during' and 'between' relations will appear only with intervals and so also, 'at' will appear only with time instants. 'Before' and 'after' may appear with both time instants and intervals.

The inference rule, [EVTM-EL1], specifies that if the **event** is true in the current world state then the event expression may be replaced by the new variable, ei-t, whose value is the time or interval of the **event**. [EVTM-EL2] indicates the role of **operator** under these conditions. **Operator** here is any operator other than ISTRUE, ISFALSE, ISUNKNOWN and OCCURS. The **operator** does not operate on the **event**, but only on the **exp**. If the **event** is not true in the current world state then the two cases are indicated by [EVTM-EL3] and [EVTM-EL4]: Here the **event** implies the **exp** with the indicated time relationships.

TABLE M-III: Event-Time Elimination Rules.

Name	Condn.	Inference.	Bindings
[EVTM-EL1]	(ISTRUE event)	[(time-rel ei-t) exp] [(time-rel (event-rel event)) exp]	(ei-t = (value-of (event-rel event)))
[EVTM-EL2]	(ISTRUE event)	[(time-rel ei-t) (operator exp)] [operator ((time-rel (event-rel event)) exp)]	(ei-t = (value-of (event-rel event)))
[EVTM-EL3]	(NOT (ISTRUE event))	([ei-t event) IMPLIES [(time-rel ei-t) exp]) ((time-rel (event-rel event)) exp)	(ei-t = (value-of (event-rel event)))
[EVTM-EL4]	(NOT (ISTRUE event))	([ei-t event) IMPLIES [operator ((time-rel ei-t) exp)]) (operator ((time-rel (event-rel event)) exp))	(ei-t = (value-of (event-rel event)))

TABLE M-IV: Time Reduction Rules.

Name	Condn.	Inference	Bindings
[TMR-U]	T	((EVERY x (tm-term range))(tm-term exp)) (tm-term ((EVERY x range) exp)))	None
[TMR-E]	T	((EXISTS x (tm-term range))(tm-term exp)) (tm-term ((EXISTS x range) exp)))	None
[TMR-N]	T	(NOT (tm-term exp)) (tm-term (NOT exp))	None
[TMR-AND]	T	((tm-term exp-1) AND (tm-term exp-2)) (tm-term (exp-1 AND exp-2))	None
[TMR-OR]	T	((tm-term exp-1) OR (tm-term exp-2)) (tm-term (exp-1 OR exp-2))	None
[TMR-IMP]	T	((tm-term exp-1) IMPLIES (tm-term exp-2)) (tm-term (exp-1 IMPLIES exp-2))	None
[TMR-TM]	(x = y)	(x exp) (x (y exp))	(x = y)
[TMR-XN]	T	(tmxn (tm-term-1 (tm-term-2 exp))) (tm-term-1 (tm-term-2 exp))	None

The binding conditions generated during the event-time rules are used by the system to order the time variables, where feasible. The expressions associated with these time variables are selected in the order of the time ordering of the variables. We will see an example of this in section 4.

Once the **event-rel** expressions are eliminated from a sequent using the [EVTM-ELi] rules, the sequent will contain only **time-rel** expressions. These are reduced using the time reduction rules, TMR-x, shown in table IV. These rules reduce complex **time-rel** expressions to combinations of simpler ones. All of them are independent of the context of their appearence in a sequent. The first four are, [TMR-U] for universally quantified expressions, [TMR-E] for existentially quantified expressions, [TMR-AND] for conjunctions and [TMR-OR] for disjunctions. As mentioned before, syntactic pattern, **tm-term**, stands for a term that specifies time.

The remaining rules shown in table IV reduce embedded timed expressions: [TMR-TM] gives the condition for replacing (x (y **exp**)) by (x **exp**), namely that (x = y), where x and y are either variables or constants (i.e. known time instants). If they are constants then the rule is applicable only if they are equal, and if x or y is an unbound variable then the binding condition specifies that they should be equal; [TMR-XN] specifies the rule for reducing arbitrary embedded time expressions. Here, **tmxn** is the function that modifies the expression to account for the intersection of the two time terms in the expression. Thus, for example, (**tmxn** ((after x)((during [y z)) exp), will be ((during [y z)) exp) if y is after x, ((during [x z)) exp) if y is not after x but z is after x, and NIL if x is after z. **tmxn** is defined for all possible combinations of time terms. Note that in general (**tmxn** (term1 (term2 exp)) is not the same as ((term1 exp) AND (term2 exp)). I will not present here the definition of this function. The rules for CREATE are presented next in tables M-Va and M-Vb.

The inference rules for CREATE may be applied to a CREATE expression even if it occurs embedded inside another larger expression. All top level CREATE expressions (those not embedded in other expressions) are retained in a sequent. The rules are applied to copies of such expressions, introduced into the sequents before the application of the rules. The system keeps a record of the expressions in a sequent which have been thus copied and expanded. This convention applies also to DESTROY and KEEP expressions.

TABLE M-Va: Reduction Rules for CREATE expressions.

Name	Condn.	Inference
[CREL-1]	(ISTRUE **exp**)	(**true-part-of exp**)
		(CREATE **exp**)
[CREL-2]	(NOT (ISTRUE **patm-exp**))	(ASSERT (**create-action-of patm-exp**))
		(CREATE **patm-exp**)
[CRR-TM]	T	(CREATE (**tm-term exp**))
		(**tm-term** (CREATE **exp**))
[CRR-DNF]	T	(CREATE (**dnf conjunction**))
		(CREATE **conjunction**)
[CRR-NOT]	T	(DESTROY **exp**)
		(CREATE (NOT **exp**))
[CRR-IMP]	T	([DESTROY **exp-1**)] OR [CREATE **exp-2**])
		(CREATE (**exp-1** IMPLIES **exp-2**))
[CRR-AND]	T	(ASSERT (.., [CREATE **exp**], ...))
		(CREATE (..AND **exp** AND ...))
[CRR-OR]	T	(..OR [CREATE **exp**] OR ...)
		(CREATE (..OR **exp** OR ...))
[CRR-OP1]	T	(**operator-a exp**)
		(CREATE (**operator-a exp**))
[CRR-OP2]	T	(**tm-term** (**operator-a exp**))
		(CREATE (**tm-term** (**operator-a exp**)))
[CRR-ASRT]	T	(ASSERT ((CREATE **exp**), ..., (CREATE **exp**)))
		(CREATE (ASSERT (**exp**, ..., **exp**)))

There are fourteen inference rules for CREATE, two elimination rules, [CREL-i], one reduction rule for timed expressions, [CRR-TM], nine for logical expressions (five for propositional combinations and four for quantifier expressions), and two for reducing operator combinations with CREATE as the first operator. If an **exp** is true in the current world state, then it has already been created. In this case, as per [CREL-1] one may simply replace the CREATE expression by (**true-part-of exp**)* in a sequent.

* The part of the **exp** that evaluates to T in the world state, and which caused the **exp** itself to have the truth value T. Thus for example, the true part of (x OR y) when x is false and y is true is y, if both are true then it is (x OR y), if x is true and y is false then it is x, else it is NIL.

TABLE M-Vb: Reduction Rules for quantified CREATE expressions.

Name	Condn.	Inference
[CRR-U →]	T	(.., [(DESTROY (ug-x ∈ **range**)) OR (CREATE (**subst** ug-x x **exp**))], ...) → ;
		(.., (CREATE ((EVERY x **range**) **exp**)), ...) → ;
[→ CRR-U]	T	→ (.., [(DESTROY (ui-c ∈ **range**)) OR (CREATE (**subst** ui-c x **exp**))], ...) ;
		→ (.., (CREATE ((EVERY x **range**) **exp**)), ...) ;
[CRR-E →]	T	(.., (ASSERT ((CREATE (ei-c ∈ **range**)), (CREATE (**subst** ei-c x **exp**)))), ...) → ;
		(.., (CREATE ((EXISTS x **range**) **exp**)), ...) → ;
[→ CRR-E]	T	→ (.., (ASSERT ((CREATE (eg-x ∈ **range**)), (CREATE (**subst** eg-x x **exp**)))), ...) ;
		→ (.., (CREATE ((EXISTS x **range**) **exp**)), ...) ;

[CREL-2] says that if the argument of CREATE is a positive atomic expression, **patm-exp**, and it is not true in the world state then it is to be replaced by the action that is needed to create it. Here, **patm-exp** can also be the name of an object or action. If it is an object then it will be replaced by the action needed to create the object.

The rule, [CRR-TM], says that time specifications may be moved into a CREATE expression, from outside. The rules [CRR-DNF], [CRR-U], [CRR-E], [CRR-AND], [CRR-IMP], [CRR-OR] and [CRR-NOT] specify methods for decomposing the creation of complex logical expressions into combinations of creation of simpler ones. The pattern, '(..AND **exp** AND ...)' in [CRR-AND] rule refers to a conjunction of expressions, where **exp** may be followed on either side by zero or more other conjuncts (thus the pattern can be simply '**exp**' itself). Each expression, **exp**, in the conjunction is replaced by '(CREATE **exp**)', inside the scope of ASSERT, and the AND connectives are replaced within the ASSERT by commas.

[CRR-AND] requires that the conjuncts in such a conjunction should all be ASSERTed jointly. This is indicated in the rule by the occurrence of the CREATE expressions nested within the outer ASSERT. Each inner CREATE expression, in this nesting will be reduced first either to ASSERT expressions or simply to atomic expressions using the appropriate logical rules and elimination rules. This will result in either a single assertion containing one or more atomic expressions, or a disjunction of such assertions. In each such assertion the atomic expressions are effectively asserted *in parallel* into the world state†. The inner CREATE expressions will get reduced as follows:

If the argument of an inner CREATE expression is a quantified expression then the quantifier rules of table M-Vb will be used until the argument is reduced to a conjunction, disjunction, a negation or an implication. If the argument is a negation then it will be changed to a DESTROY expression, using [CRR-NOT]. If the argument is an implication then [CRR-IMP] will be used. If it is a disjunction then [CRR-OR] will be used. If it is a conjunction then it will be first put in *disjunctive normal form* (dnf). This is done by using the [CRR-DNF] rule. The disjunctive normal form expression is a disjunction of conjunctions. Thus [CRR-OR] rule may be used after [CRR-DNF] to separate out the disjunctions. This process is iterated until one gets CREATE expressions consisting only of conjunctions of atomic expressions. In this process it may often be necessary to employ the [CRR-OPi], [ASRT-ASRT], [ASRT-OP] and [ASRT-OR] rules in table M-VIIIb, to get rid of certain nested CREATE (and ASSERT) expressions. [CRR-OPi], specifies that the creation of an **operator-a** expression is the same as the **operator-a** expression itself, where **operator-a** is the same as **operator**, but excluding ASSERT. The [ASRT-ASRT] specifies that inner ASSERTs in nested ASSERT expressions may be reduced to the arguments of the ASSERT expressions.

Table M-Vb shows the reduction rules for quantified expressions. The pattern (.., quantifier-exp, ...) indicates a form that might appear inside an ASSERT. I have not shown the ASSERT itself in table M-Vb, because I wanted to indicate that the same rules apply also to the case where '(CREATE quantified-exp)' appears alone in the sequent. The expressions '(.. ∈ **range**)' in this table are the *binding conditions*. These conditions indicate the ranges of the variables that are newly generated by the application of these rules. Notice that the top expression in the inference of [CRR-U →] may be viewed as being the result of application of ([U →], [CRR-IMP], [OR →]), in the left to right order, to the expression at the bottom of the inference. The only difference is

† The problems associated with the monitoring of parallel actions have not been adequately investigated yet. In most cases they require specialized routines to monitor parallel assertions of actions and their behaviors. No general principles of organization has emerged yet. I expect to gain some understanding for the nature of these problems from the current application domain, namely Naval Operational Planning. The inference rules presented in this section show the machinery used in the inference engine to accommodate parallel monitoring of multiple actions.

that [CRR-U →] allows one to apply [U →] to the universally quantified expressions that occur inside the scope of a CREATE. The other rules in table M-Vb may be similarly interpreted. These are the *quantifier elimination* rules for CREATE expressions.

The variables ug-x and eg-x in the rules [CRR-U →] and [→ CRR-E] are the *generalization variables* mentioned previously. These variables are used to represent *arbitrarily selected* items from their respective **ranges**. They *must be new variables, not previously used in the deduction tree.* Similarly, ei-c and ui-c are *existential* and *universal instantiation* variables. These denote distinct constants that are (or will be) *newly instantiated* in the inference process.

The inference rules in tables M-Va and M-Vb together specify the means for eliminating the CREATE operators from a sequent. The creation of a complex logical expression is decomposed to the creation of its simpler components, and the creation of the simplest unit is reduced to the actions that are needed to create it. The important point to note here is that, in general ([CREATE x] AND [CREATE y]) is not the same as (CREATE (x AND y)) for expressions x and y. This is because, even though [CREATE x] and [CREATE y] may both be true, taken separately, their joint creation may not be always true. Thus, for example, one may be able to buy a car and buy a house, but may not have the necessary resources to buy them both.

Similar rules for the DESTROY operation are shown in tables M-VIa and M-VIb. The rules here are the dual of the rules in CREATE. For an arbitrary **exp** (that is not a positive atomic expression), if the **exp** is true in the current world state, then the **true-part-of** the **exp** is to be destroyed. For a **patm-exp**, if it is true in the world state then the **destroy-action-of** the **patm-exp** replaces the DESTROY expression. If **patm-exp** is an object then it is replaced by the actions needed to destroy the object. It may be noted that (DESTROY (x OR y)) is not the same as ((DESTROY x) OR (DESTROY y)), instead it is (ASSERT ((DESTROY x) AND (DESTROY y))). This indicates that the actions needed to (DESTROY x) and (DESTROY y) should be created in parallel.

The destruction of a conjunction, is the same as the conjunction of destructions. This also is the dual of the situation that occurred for the CREATE expressions, where the creation of a disjunction was the same as the disjunction of creations. For DESTROY expressions, using rule [DES-CNF], its argument is put in *conjunctive normal*

TABLE M-VIa: Reduction Rules for DESTROY expressions.

Name	Condition	Inference
[DES-EL1]	(ISFALSE **exp**)	(NOT **exp**) (DESTROY **exp**)
[DES-EL2]	(ISTRUE **patm-exp**))	(ASSERT (**destroy-action-of patm-exp**)) (DESTROY **patm-exp**)
[DES-RED]	(ISTRUE **exp**)	(DESTROY (**true-part-of exp**)) (DESTROY **exp**)
[DES-TRM]	T	(DESTROY (**value-of term**)) (DESTROY **term**)
[DES-TM]	T	(DESTROY (**tm-term exp**)) (**tm-term** (DESTROY **exp**))
[DES-CNF]	T	(DESTROY (**cnf disjunction**)) (DESTROY **disjunction**)
[DES-NOT]	T	(CREATE **exp**) (DESTROY (NOT **exp**))
[DES-IMP]	T	(ASSERT ([CREATE **exp-1**], [DESTROY **exp-2**])) (DESTROY (**exp-1** IMPLIES **exp2**))
[DES-AND]	T	([DESTROY **exp-1**] OR [DESTROY **exp-2**])) (DESTROY (**exp-1** AND **exp-2**))
[DES-OR]	T	(ASSERT (.., (DESTROY **exp**), ...)) (DESTROY (..OR **exp** OR ...))
[DES-ASRT]	T	((DESTROY **exp**) AND ... AND (DESTROY **exp**)) (DESTROY (ASSERT (**exp**, ..., **exp**)))

form, (a conjunction of disjunctions). The [DES-OR] rule is always applied simultaneously to all the disjuncts in a disjunction.

The rule for [DES-U →] may be interpreted as follows:

[2.8]. (DESTROY ((EVERY x **range**) **exp**)) IFF
[2.9]. (DESTROY ((EVERY x)([x ∈ **range**] IMPLIES **exp**)) IFF
[2.10]. ((EXISTS x)(DESTROY ([x ∈ **range**] IMPLIES **exp**))).

One gets [DES-U →] by applying ([E →], [DES-IMP]) in sequence to the expression (4.3)

TABLE M-VIb: Reduction Rules for quantified DESTROY expressions.

Name	Condition.	Inference
[DES-U →]	T	(ASSERT (.., ([CREATE (ei-c ∈ **range**)], [DESTROY [**subst** ei-c x **exp**])), ...) → ;
		(.., (DESTROY ((EVERY x **range**) **exp**)), ...) → ;
[→ DES-U]	T	→ (.., (ASSERT ([DESTROY (**subst** eg-x x **exp**)], [CREATE (eg-x ∈ **range**)])), ...) ;
		→ (.., (DESTROY ((EVERY x **range**) **exp**)), ...) ;
[DES-E →]	T	(.., ([DESTROY (**subst** ug-x x **exp**)] AND [DESTROY (ug-x ∈ **range**)]), ...) → ;
		(.., (DESTROY ((EXISTS x **range**) **exp**)), ...) → ;
[→ DES-E]	T	→ (.., ([DESTROY (**subst** ui-c x **exp**)] AND (DESTROY (ui-c ∈ **range**)), ...) ;
		→ (.., (DESTROY ((EXISTS x **range**) **exp**)), ...) ;

above, and [→ DES-U] by applying ([→ E], [DES-IMP]) to [2.10]**. Similarly,

[2.11]. (DESTROY ((EXISTS x **range**) **exp**)) IFF
[2.12]. ((EVERY x)(DESTROY ([x ∈ **range**] AND **exp**))),

and the [DES-E →] and [→ DES-E] rules are derived from the expression [2.12] by applying rules ([U →], [DES-AND]) and ([→ U], [DES-AND]) respectively.

Notice that the rules in table M-VIa uses an ISFALSE condition on [DES-EL1] and ISTRUE condition on [DES-EL2] and [DES-RED]. Both ISFALSE and ISTRUE are false then it would mean that the truth of the arguments of DESTROY is unknown in the world. In this case CK-LOG will prompt the user to specify whether the truth of the expression should be hypothesized. This kind of interaction between the TPS and the world state models is used in CK-LOG to identify information that is unknown but is needed for the solution of a problem. This feature is extensively used in the example discussed in [Srinivasan 1984], leading at times to the creation of intelligence gathering operations to get the information.

** The rules [E →], [→ E], [OR →], [→ OR], [→ AND], [AND →], [U →], and [→ U] were introduced previously in this section in tables 2-I and 2-II.

TABLE M-VII: Reduction Rules for PREVENT.

Name	Condition	Inference
[PRR]	T	([DESTROY **sexp**] AND [PREVENT (CREATE **sexp**)])
		(PREVENT **sexp**)
[PRR-TRM]	T	(PREVENT (**value-of term**))
		(PREVENT **term**)
[PRR-TM]	T	(PREVENT (**tm-term exp**))
		(**tm-term** (PREVENT **exp**))
[PRR-AB]	T	([DESTROY ((**before tm-term**) **sexp**)] AND [PREVENT (CREATE ((**aft-bef tm-term**) **sexp**)])
		(PREVENT ((**aft-bef tm-term**) **atm-exp**))
[PRR-INT]	T	([DESTROY ((before **tm-term-1**) **sexp**)] AND [PREVENT (CREATE ((before **tm-term-2**) **sexp**))])
		(PREVENT ((**dur-bet** (**tm-term-1 tm-term-2**)) **sexp**))

The reduction rules for PREVENT are presented next in table M-VII. Preventing an expression from becoming true is the same as first destroying its truth and then preventing its recreation. For simple expressions (i.e. expressions without any time qualifiers), **sexp**, this is expressed by the rule [PRR] in table M-VII. If the argument of a PREVENT expression is a term then the intention is to prevent the truth of the value of the term. This is expressed by the rule [PRR-TRM] in table M-VII. The rule [PRR-TM] specifies that **tm-terms** may be moved inside the scope of PREVENT from outside. The reduction rules for timed expressions are stated next in table M-VII in rules [PRR-AB] (for after, before expressions) and [PRR-INT] (for between and during expressions). The pattern, **aft-bef** in [PRR-AB] will match 'after' or 'before', and **dur-bet** will match 'during' or 'between'. In all cases the PREVENT expression gets transformed to a conjunction of a DESTROY expression and the PREVENTion of its recreation. Again note that (PREVENT (x OR y)) will not be the same as ((PREVENT x) OR (PREVENT y)).

The important point to note is that PREVENT has a semantics that is quite different from those of CREATE and DESTROY. Whereas CREATE and DESTROY are one shot operations, PREVENT requires a persistent action; there is need to prevent the recreation of the destroyed object. Also note that (PREVENT (NOT exp)) will get transformed to

([DESTROY (NOT exp)] AND [PREVENT (CREATE (NOT exp))])

by the [PRR] rule, and this in turn will get transformed to

([CREATE exp] AND [PREVENT (DESTROY exp)])

via rules ([CRR-NOT], [DES-NOT]).

Similarly, there are inference rules for other operators as well: Rules for KEEP, SUPPORT, etc. I will not present here the rules for these. Let me conclude this section with the inference rules for ASSERT. The ASSERT operator is used to make an expression true in a world state without having to go through its associated CREATE processes. The components of the statement are simply asserted into the world state. As mentioned previously, the assertion will occur unconditionally if the ASSERT statement occurs on the left side of a sequent, and conditionally if it is on the right.

TABLE M-VIIIa: Reduction Rules for ASSERT.

Name	Condn.	Inference and Bindings.
[ASRT-TM]	T	(ASSERT **exp**) binding: (ei-t = **tm-term**).
		(**tm-term** (ASSERT **exp**))
[ASRT-TRM]	T	(ASSERT (**value-of term**))
		(ASSERT **term**)
[ASRT-ASRT]	T	(ASSERT (..., **exp**, --, **exp**, ..))
		(ASSERT (..., (ASSERT (**exp**, --, **exp**)), ..))
[ASRT-OP]	T	(**operator-cd** (**exp** AND ... AND **exp**))
		(**operator-cd** (ASSERT (**exp**, ..., **exp**)))
[ASRT-EL]	(ISTRUE **exp**)	[(**true-part-of exp**) OR (ASSERT (**not-true-part-of exp**))]
		(ASSERT **exp**)

The first five rules in table M-VIIIa are quite straight forward. [ASRT-TM] in effect says that the time at which the assertion was made may be ignored in the reasoning process. The time of the assertion is however saved in the binding condition in a

TABLE M-VIIIb: Reduction Rules for ASSERT (contd).

In the following the condition 'not-true' = '(NOT (ISTRUE exp)),' where exp is the conjunction of the arguments of ASSERT.		
[ASRT-EL →]	not-true	(**L-assert** (**atm-exp**, ..., **atm-exp**)) → ;
		(ASSERT (**atm-exp**, ..., **atm-exp**)) → ;
[→ ASRT-EL]	not-true	(**R-assert** (**atm-exp**, ..., **atm-exp**)) → ;
		→ (ASSERT (**atm-exp**, ..., **atm-exp**)) ;
[ASRT-DNF]	not-true	(ASSERT (**dnf conjunction**))
		(ASSERT **conjunction**)
[ASRT-U →]	not-true	(ASSERT (.., [(ug-x ∈ **range**) IMPLIES (**subst** ug-x x **exp**)], ...)) → ;
		(ASSERT (.., ((EVERY x **range**) **exp**), ...)) → ;
[→ ASRT-U]	not-true	→ (ASSERT (.., [(ui-c ∈ **range**) IMPLIES (**subst** ui-c x **exp**)], ...)) ;
		→ (ASSERT (.., ((EVERY x **range**) **exp**), ...)) ;
[ASRT-E →]	not-true	(ASSERT (.., (**subst** ei-c x exp)), (ei-c ∈ **range**), ...)) →;
		(ASSERT (.., ((EXISTS x **range**) **exp**), ...)) → ;
[→ ASRT-E]	not-true	→ (ASSERT (.., (**subst** eg-x x exp)), (eg-x ∈ **range**), ...)) ;
		→ (ASSERT (.., ((EXISTS x **range**) **exp**), ...)) ;
[ASRT-OR]	not-true	([ASSERT **exp-1**] OR [ASSERT **exp-2**])
		[ASSERT (**exp-1** OR **exp-2**)]
[ASRT-AND]	not-true	(ASSERT (.., **exp**, ...))
		(ASSERT(..AND **exp** AND ...))
[ASRT-IMP]	not-true	(ASSERT (.., ([NOT **exp-1**] OR **exp-2**), ...))
		(ASSERT (.., (**exp-1** IMPLIES **exp-2**), ..))

newly generated local variable, ei-t, for future use if necessary. Since we are not interested here in analyzing who said what and when, I have chosen this simplification. In [ASRT-EL] the **not-true-part-of** function will return the part of **exp** that is not true in the world state. This is the dual of the **true-part-of** function mentioned earlier. The rest of the rules in the table need some explanation.

In its simplest form, the argument of an ASSERT expression is a series of expressions separated by commas. As indicated by the [ASRT-AND] these commas are interpreted within the ASSERT as AND . Inside an ASSERT, the AND 's may be replaced by commas. This is called a parallel ASSERT. If all the arguments of an ASSERT are atomic expressions then for each atomic expression its truth value is first set in the world state to T if it is positive, and to F if it is negated. Only after setting the truth values for all the atomic expressions in the ASSERT statement will CK-LOG check the world state for consistency. Thus, in effect, they are all asserted in parallel. This is what happens when the **L-assert** function is executed in [ASRT-EL →] rule. The **L-assert** function will return the conjunction of all the atomic expressions that were successfuly asserted into the world state. It will return NIL if the assertion is not successful. The **L-assert** happens unconditionally, when the ASSERT statement is on the left side of a sequent. As mentioned before an assertion may cause the system to augment the deduction tree with hypotheses problems, or may get rejected.

If it is on the right side of a sequent, then the **R-assert** function is used. This function will first check whether the arguments of the ASSERT expression satisfy the matching requirements described earlier. If all the atoms in the ASSERT expression are matched successfuly then the assertion will be performed using the **L-assert** function. If matches could not be found for some or all of the arguments of the assertion, then the system will present to the user the list of atomic expressions for which matches could not be found in the sequent. At this point the user may either advise the system to reject the assertion, or ask the system to accept the atoms even though no matches exist for them in the sequent. The conjunction of these atoms will then be introduced by the system as a *hypothesis problem* associated with the sequent in which the ASSERT expression appeared.

[ASRT-DNF] puts the argument of an ASSERT into disjunctive normal form, if the argument is a **conjunction** (which can be a series of expressions separated by commas). Notice that in the rules above the pattern '(..,exp, ...)' indicates one or more expressions, exp, separated by commas (or AND , or OR as the case may be). [ASRT-OR] converts the ASSERT of a disjunction to a disjunction of ASSERTS. If the argument is quantified expression, then ASRT-U and ASRT-E are used to eliminate the quantifiers. They introduce the new variables, ug-x, ui-c, eg-x and ei-c depending on the various cases. The binding conditions specify their ranges and indicate that these were created inside the scope of an ASSERT statement. These rules allow the quantifier elimination rules to be applied even when the quantifiers are inside the scope of an ASSERT. In [ASRT-OP] rule the pattern, **operator-ed** is the same as **operator** with CREATE and DESTROY excluded. I have assumed throughout that negations will appear only with atomic expressions.

Among the rules presented above and those presented in tables 2-I and 2-II, the various rules for NOT, AND, OR, IMP and IFF expressions, and the rules for reducing timed expressions, are called ***propositional rules***. The rules for quantified expressions are called ***quantifier elimination rules***. In applying a rule to a given sequent there two choices to be made: The expression in the sequent to which a rule is to be applied, and the rule to be applied to the chosen expression. The choice of an expression in a sequent might in certain cases be determined by the ordering of time expressions associated with them. TMS has special facilities to order time instants over an after/before lattice. If there are several choices available for expressions, then either one may be chosen arbitrarily or the user may specify to the system the choice to be made.

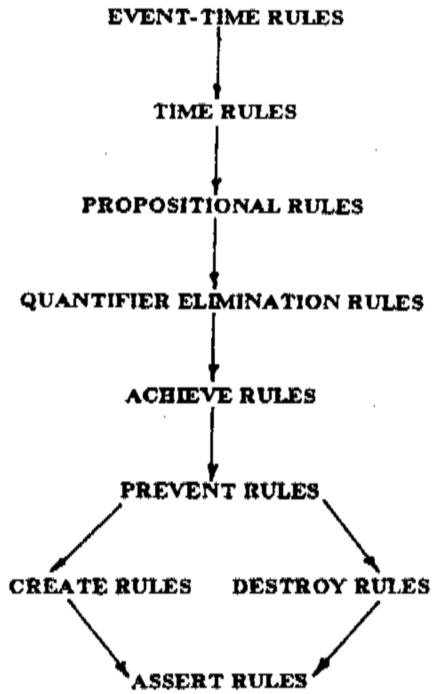

Figure 2.3: The Order of Application of Inference Rules.

Once the expression is chosen the rule (set of rules) to be applied to it will usually be unique. These rules are applied to the chosen expression in the following order: First all the applicable event-time rules (table M-III) are applied, then all the applicable time rules (table M-IV) are applied, then the applicable propositional rules (table 2-I) are applied, then the quantifier elimination rules (table 2-II), then the ACHIEVE rule (table M-I), then the PREVENT rules (care should be taken here to prevent endless loop because of recursion in the PREVENT rules), then the CREATE or DESTROY rules, and finally the ASSERT rules. This is diagramed in figure 2.3.

In the last three subsections I have presented several modifications to the mating algorithm presented in section 2.5. The effects of the modifications on the completeness and consistency of the theorem proving system, TPS, has not been completely analyzed yet. The ASSERTion semantics discussed in section 2.7 does not by itself affect the completeness and consistency of the TPS. However, the possibility of user introduced assertions in the middle of a theorem proving process may have unforeseen consequences. If there are inconsistencies in the user given assertions, then they will ultimately be detected by TMS as TPS continues with an expanding deduction tree.

The way the inference rules for other modal operators affect the completeness and consistency of TPS would depend strongly on the kind of action predicates that one uses to monitor and follow the effects of actions in TPS, the way the *function* and *behavior* of actions have been defined, and the facilities available for monitoring parallel actions. At present I have no guidelines to offer a user on defining these. More experience with the use of the system is necessary before one can comment on this.

As I mentioned earlier, the interaction between the world states and TPS is used to identify and use unknown information in the world state as hypotheses. But if **K**[U] itself is incomplete then there is no way of generally predicting what might happen. In section 5 I introduce the concept of *adequacy* of **K**[U] and propose the *locality condition* as a sufficient condition for its adequacy.

3. MDS, The Meta Description System.

3.1. An Example of Concept Definition.

Figure 3.1 shows a typical concept definition in CK-LOG*. But for some minor syntactic differences this is the way one would use **TML** in CK-LOG to define concepts. The concept defined here is called MIL-ACTION (military action). MIL-ACTION is a concept in the sense that it denotes the set of all possible military actions. Each structure definition in the figure introduces one or more component concepts of MIL-ACTION, and possibly also a *relation name* that is used to indicate the relationship between the parent concept and the component(s). The meaning of each of these relations is defined to CK-LOG in two ways: The first is by the specification of *consistency conditions* (section 3.4) which the relations should satisfy in a world state. The second is by the specification of the actions (section 3.5) needed to make these relations true in a world state (the **create-action** for the relation), to maintain its truth (the **keep-action** for the relation), or to destroy its truth (the **destroy-action** for the relation). Let me briefly elaborate on the significance of the definitions shown in this figure.

3.2. The function, behavior, analysis and design of MIL-ACTION.

Let me begin with the *function, behavior, design* and *analysis* of MIL-ACTION. The *function* in figure 3.1 uses the *operators*, ACHIEVE, ISTRUE and PREVENT. Let me briefly explain what these operators refer to and what they mean (the inference rules for these were discussed in section 2.9). (ACHIEVE (execution-of mil-action)) refers to the set of processes that may be invoked to perform the said executions. It will be true in a world state when all the processes involved in the execution terminate successfuly in that world state. (ISTRUE (objective-of mil-action)) declares that the objective of the action should be true in a world state. (PREVENT (purpose-of mil-action)) refers to the processes that might prevent the achievement of the purpose-of the mil-action, and (NOT (OCCUR (PREVENT ...))) says that these processes should not occur†. The expression '(*time-of* (ACHIEVE (execution-of mil-action)))' refers to the time instant when the

* The example presented here is taken from the OPPLAN-CONSULTANT, [Srinivasan 1984]. I am using this example here because it illustrates details of knowledge definition not covered by the example discussed in section 4.

† 'OCCURS' and 'OCCUR' are synonymous words. I am using both of them here for better readability.

(ACHIEVE (execution-of ...)) becomes true.

The *function* in figure 3.1 may now be paraphrased as follows: At the time the execution-of the mil-action is achieved (i.e. completed successfuly), its objective should be true and its purpose should not be prevented.** Notice that this *function* definition does not specify when the execution itself should be performed. This is specified in the OPPLAN definition: the execution of a mil-action will commence only when the status-of its OPPLAN becomes active. CK-LOG will use *function* statements like this to set up goals for itself in its problem solving activities. When a mil-action is instantiated CK-LOG will update the left sides of the relevant sequents in the frontier set of its deduction tree with the action predicates,

[3.1]. [(before (*time-of* (status-of mil-action successful)))
(**value-of** (*function* mil-action))],
[(before (*time-of* (**value-of** (*function* mil-action)))
(**value-of** (*behavior* mil-action))],

to achieve the *function* and *behavior* of the action in the existing world state. If CK-LOG is not able to perform the necessary modifications in the world state, then there is something wrong with the action, either it should not be terminated or something else should be done depending on what the user decides.

Function and *behavior* (as also the *analysis* and *design*) definitions like these may also appear with concepts that are not actions. Thus one may have a *function* for the concept OPORDER, an operational order. Its function might say that the commander receiving the oporder should start planning for the military actions assigned to him soon after he receives the oporder. In this case the *function* of an oporder will be invoked to modify the appropriate sequents in the frontier set of a deduction tree, everytime a new OPORDER is created in a world state, and the analysis of this function would initiate the necessary planning activity.

The *behavior* of a MIL-ACTION shown in figure 3.1 makes a general statement about what happens in a mil-action. Here int is a time INTERVAL that is specified by a pair of

** It should be noted that an expression of the form, '[(*time-of* x) y]' (or '[(after (*time-of* x)) y]') where x and y are logical expressions, establishes a ***causal connection*** between the statements x and y: Making x true should cause y to become true at the same time (or after that time). In any such causal connection the logical assertion '[x IMPLIES y]' will be true. But, the causal assertion makes a stronger statement than the implicational assertion, because it also specifies a time relationship.

Structure:	(MIL-ACTION FORCE FORCE REGION) (prosecuted-by MIL-ACTION FORCE), 1 (is-against MIL-ACTION FORCE), 1 (region-of MIL-ACTION REGION), 1 (ordered-by MIL-ACTION OPORDER), 1 (specified-by MIL-ACTION OPORDER-TASK), 1 (objective-of MIL-ACTION MIL-OBJECTIVE), 1 (purpose-of MIL-ACTION MIL-OBJECTIVE), 1 (mil-plan-for MIL-ACTION OPPLAN), 1 (commander-of MIL-ACTION COMMANDER), 1 (phy-objectives-of MIL-ACTION OBJECTs), 1 (supports MIL-ACTION MIL-ACTIONs), *Irreflexive* (opposes MIL-ACTION MIL-ACTIONs), *Irreflexive,* (execution-of MIL-ACTION MILA-EXECUTION), 1 (status-of MIL-ACTION MILA-STATUS), 1 (risk-factor-of MIL-ACTION RISK-FACTOR), 1
Function:	[(*time-of* (ACHIEVE (execution-of mil-action))) ([ISTRUE (objective-of mil-action)] AND [NOT (OCCUR (PREVENT (purpose-of mil-action)))])]
Behavior:	[(EXISTS int INTERVAL) [(((during int)(MIL-ACTION x y region)) IMPLIES ([(EXISTS force FORCE) ((between int) ([OCCURS (DESTROY force)] AND [(belongs-to force x) OR (belongs-to force y)]))] OR [(EXISTS r1,r2 (is-within region)) ((between int) ((controls x r1) AND (controls y r2)))]) AND ((during int) (OCCURS (ACHIEVE (execution-of mil-action))))]]
Analysis:	[ACHIEVE (execution-of mil-action)]
Design:	[ACHIEVE (*design* (mil-plan-for mil-action))]

Figure 3.1: The Concept of Military Action, MIL-ACTION.

TIME points, (t_1 , t_2). The statement says the following: If there is an int during which (MIL-ACTION x y region) is true, for any x, y and region, then there are DESTROY processes against FORCEs belonging to x and y that will OCCUR between the time points of the same int, or there will exists REGIONs within region that are controlled by the participating forces x and y between the same time points, or both of these will occur (OR is inclusive or). The expression, '(during int)' is interpreted as 'for all time instants, t, in the semiclosed interval, $[t_1, t_2)$', and '(between int)' is interpreted as 'there exists time instants, t, in the open interval, (t_1, t_2)'.

The DESTROY operator may operate on instances of concepts, like force above, or on logical expressions containing particular or generic instances of concepts. When it operates on an instance, it refers to the processes, P, that might be needed to destroy, i.e. make false, ***all*** or an ***a priori specified subset*** of the properties of the instance. When it operates on a logical expression, it refers to the processes, P, that might be needed to make the expression false. It will be true (become true) in a world state if the specified properties are (become) false in that world state. If the specified properties are not false and P is not NIL then the truth value of (DESTROY ...) will be ? and that of (OCCURS (DESTROY ...)) will be T, because the processes, P, are occurring. If P is NIL then (OCCURS (DESTROY ...)) is F. If the specified properties are not false and P is NIL then the truth value of (DESTROY ...) is F. Notice that (DESTROY ...) may be true in a world state (because the specified properties are false) while (OCCURS(DESTROY ...)) is false (because P is NIL). Similar considerations apply also to CREATE, ACHIEVE and other operators used in this paper; these will attempt to make the specified properties true. I will say more on the OCCURS operator in section 3.4.

It may be noted that the behavior statement in figure 3.1 does not say that there should necessarily be destruction of FORCEs: (OCCURS (DESTROY force)) simply says that DESTROY processes against force occurs. The DESTROY processes may not succeed: If the processes started by DESTROY all terminate without making the specified properties false then destruction would not have occurred. The use of 'between' instead of 'during' indicates that the DESTROY processes need not occur all through the int. Similarly, the use of 'between', in the statement about regions controlled, allows the regions controlled to change at different times in the int. Thus the behavior statement is a very general statement that captures the nature of a MIL-ACTION: In all world states, if a military action occurs then this is what one may expect.

Behavior statements like this are used in CK-LOG to analyze the behavior of actions in a world state. In a planning environment these may be used to anticipate the contingencies which should be accounted for in its plans, or to plan the destruction of an action. The above *behavior* statement may thus be used by the system to set up planning subgoals to protect a commander's forces as a part of a military planning process, since it knows that every commander has to keep his forces operational and to keep them operational their destruction should be prevented. The standard way of destroying an action in CK-LOG is by destroying its ***behavior***, i.e. by making its ***behavior*** false in a world state. As mentioned before, everytime a new instance of an ACTION, say action, is created in a world state, the ***behavior*** and ***function*** of the action are used to update the goal set in a problem solving process.

The ***analysis*** definition declares that a military action is analyzed by analyzing the way it achieves its execution. Thus, to analyze a mil-action one would set up the subgoal,

[3.2]. $\rightarrow$ (ACHIEVE (execution-of mil-action)) ;

or the inference engine will transform the subgoal,

[3.3]. $\rightarrow$ (ACHIEVE (*analysis* mil-action)) ;

to subgoal [3.2] above using the ***analysis*** definition of MIL-ACTION. Similarly, to design a military action the plan for the action should be designed (see [Srinivasan 1984] for details on how plans are designed). Let me now present the interpretations given to the ***structure*** definition shown in figure 3.1.

3.3. The structure of MIL-ACTION.

The first structure definition in figure 3.1 is the tuple,

(MIL-ACTION FORCE FORCE REGION).

This declares that a MIL-ACTION always involves two FORCEs and a REGION, where FORCE and REGION are also concepts in the universe U. These are the components that give a MIL-ACTION its unique identity: two military actions are different if and only if at least one of these components is different for them. Having made this declaration one may now use in CK-LOG expressions of the form,

(**t-exp** (MIL-ACTION $force_1$ $force_2$ region)), to denote the occurrence of a military action between the particular FORCEs, $force_1$ and $force_2$ in the REGION, region, at the time instant (or time interval) specified by the time-expression, **t-exp**. The entire expression shown above is a proposition in the language **DL**, i.e. it will have a truth value, T (*true*), F (*false*) or ? (*unknown*), in world states, U_t. I will refer to expressions of this kind, with or without negation, as *timed literals**. If the truth value of the above timed literal is T then it would mean that a military action between the said forces is occurring at the time in the said region, if it is F then it is not occurring, and if it is ? then it is not known whether it is occurring or not. If no time parameter is given then the expression refers to its truth value in the *current world state*. Notice that the concept MIL-ACTION has been used here also as the name of the predicate that refers to its occurrence. This is a standard practice followed for ACTIONs in CK-LOG.†

A **t-exp** is either a time instant, t, or an interval, (t_1 t_2), or an expression of the form (at **t-exp**), (after **t-exp**), (before **t-exp**), (during **t-exp**), (between **t-exp**), (*time-of event*), or (*interval-of event*).

The remaining parts of the structure definition enumerate the components of a MIL-ACTION and show how they are related to it. Thus for example, the structure

(prosecuted-by MIL-ACTION FORCE)

declares that FORCE is a component of MIL-ACTION, and it is related to MIL-ACTION via the relation name, 'prosecuted-by'. In CK-LOG, this specification is interpreted as saying that a MIL-ACTION is prosecuted-by *at most* one FORCE. However, we need to specify that there should be a unique FORCE. This is done by the declaration,

(prosecuted-by MIL-ACTION FORCE), 1

shown in figure 3.1, where a '1' appears after the comma. The '1' here is interpreted as

* A negated timed literal will have the form, '(NOT (t-exp ...))'. A literal or a timed literal may also have variables in them instead of constants as shown above.

† This may be used for any concept. The tuple associated with a concept will specify the sufficiency condition for two instances of the concept, at the same time instant, to be distinct.

specifying that there should be *at least* one FORCE. These two restrictions together imply that there should be *exactly* one such force.

Having made this declaration, one may use the term, '(prosecuted-by MIL-ACTION)', in the language **DL** to denote the concept, FORCE:

(prosecuted-by MIL-ACTION) = {FORCE}.

For a particular MIL-ACTION, say mil-action and a particular FORCE, say force, the timed literal

(**t-exp** (prosecuted-by mil-action force)),

is an atomic proposition in the language **DL**. Its truth value in a world state U_t will be T (if it is *true*), F (if it is *false*) or ? (if it is *unknown*). Also,

(prosecuted-by mil-action) = {force},

where the force is such that the truth value of '(prosecuted-by mil-action force)' is T or ? in the *current* world state. If the truth value is ? then there can be more than one such force in the current world state for which the ? truth value applies. Then '(prosecuted-by mil-action)' will denote the set of all such FORCEs. This set may be interpreted as the set of possible candidate FORCEs, one of which may be the force that prosecutes the mil-action. If the truth value is F for all FORCEs in a world state, then (prosecuted-by mil-action) will be equal to NIL (the *empty set*).

Please note that throughout this paper I will use upper case words like MIL-ACTION, FORCE, etc. to denote concepts, lower case words for names of relations and lower case small letter words like mil-action, force, etc., with or without subscripts, to denote particular (or generic) instances of concepts.

The remaining structure declarations in figure 3.1 similarly define the terms shown in figure 3.2, where OPORDER stands for the concept of an Operational Order, OPPLAN is an Operational Plan, and OPORDER-TASK is a task specified by the OPORDER, etc. The MIL-OBJECTIVEs that specify the objective and purpose** of a military action will be

** The objective and purpose can be different from each other. Thus for example the objective might be 'Seize and occupy the Xray area on the southern shore of Europe' and its purpose might be to 'support allied operations in Europe.' (Example taken from the Naval Warfare Publication, NWP-11).

(is-against MIL-ACTION)	= {FORCE},
(region-of MIL-ACTION)	= {REGION},
(ordered-by MIL-ACTION)	= {OPORDER},
(mil-plan-for MIL-ACTION)	= {OPPLAN},
(commander-of MIL-ACTION)	= {COMMANDER},
(specified-by MIL-ACTION)	= {OPORDER-TASK},
(objective-of MIL-ACTION)	= {MIL-OBJECTIVE},
(purpose-of MIL-ACTION)	= {MIL-OBJECTIVE},
(status-of MIL-ACTION)	= {MILA-STATUS},
(risk-factor-of MIL-ACTION)	= {RISK-FACTOR}, etc.

Figure 3.2: Terms defined by the structure of MIL-ACTION.

given by this oporder-task, as also the forces that prosecute the military action. The MILA-STATUS will specify the status of the military action: whether it is ongoing or not, if it is then the losses and casualities encountered by own and enemy forces in the action, the region controlled by the forces, the well-defended regions, regions under attack, etc., and if it is not then its readiness, etc. The RISK-FACTOR will specify the risk involved in the action and the chances for destroying the enemy forces. It will depend on whether the region of the action is well-defended by the enemy or not. I will not present here the detailed structures that are used to represent these concepts (See [Srinivasan 1984] for details).

Notice that I have used the plural form, OBJECTs, in figure 3.1 in the structure '(phy-objectives-of MIL-ACTION OBJECTs)'. This indicates that there could be more than one OBJECT, x, y, z, etc., in a world state such that

(phy-objectives-of mil-action x),
(phy-objectives-of mil-action y),
(phy-objectives-of mil-action z), etc

are all true for a military action, mil-action, in that world state. The declaration,

(phy-objectives-of MIL-ACTION OBJECTs), 1

in the figure specifies that there should be *at least* one physical objective for each military action. In general, in situations like this, one may associate a pair of integers, (m,n), with the structure, where m is the *lower bound* on the structure, and n is the *upper bound,* i.e. at least m and at most n objects.

A MIL-ACTION may support more than one MIL-ACTION, and the term (supports mil-action) will refer to the set of all MIL-ACTIONs supported by mil-action in a world state. The restriction that a MIL-ACTION can support only other MIL-ACTIONs that are different from itself is declared by associating the flag, *irreflexive* with the structure as in*,

(supports MIL-ACTION MIL-ACTIONs), *Irreflexive*

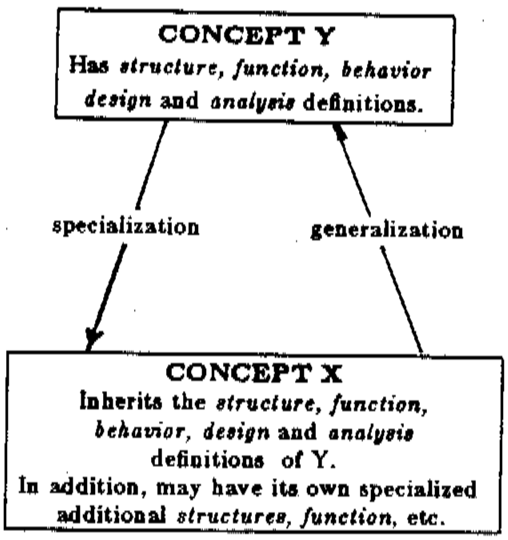

Figure 3.3: Concept of Specialization and Generalization.

One may have various kinds of MIL-ACTIONs. It can be a DESTROY-ACTION, SUPPORT--ACTION, BLOCKADE-ACTION, CAPTURE-ACTION, TRANSPORT-ACTION,, etc., or a NAVAL-ACTION, AIR-ACTION, etc. These are the *specializations* of MIL-ACTION. In CK-LOG a specialization, X, of a concept, Y, will inherit from Y all of its *structure*, and share with Y its *function, behavior, analysis* and *design* aspects. In addition X may also have its own specialized additional *structure, function*, etc. There are also facilities to

* *Symmetric, anti-symmetric, transitive* and *reflexive* are other flags of this kind that one may associate with a structure. There are also flags that one may use to control the properties inherited by a structure from its generalizations, flags that indicate the permanence of a relation, and flags that cause the system to prompt a user for values under certain conditions.

specify exceptions to this general inheritance rule. Figure 3.3 illustrates the normative rules for inheritance over the generalization (specialization) hierarchy.

The relational forms,

(RELATION-NAME CONCEPT_1 CONCEPT_2), and
(TUPLE-NAME CONCEPT_1 ... CONCEPT_k)

used in *structure* definitions are called *dimensions* in CK-LOG. In choosing this name, I seek to suggest an analogy between forms like

(parent-of INFANT PERSON),
(commander-of MIL-ACTION COMMANDER), etc.,

and forms like

(weight-of OBJECT POUNDS),
(length-of OBJECT FEET),
(area-of REGION SQUARE-MILES), etc.,

which are normally viewed as specifying the dimensions of the indicated measurements. It is dimensionally inconsistent to say, '(length-of OBJECT POUNDS)' or '(area-of REGION FEET)'. Similarly, one might say that forms like '(parent-of INFANT MIL-ACTION)', '(commander-of MIL-ACTION INFANT)', etc., are also dimensionally inconsistent in our universe.

If the relational form '(r X Y)', or the tuple form (X Y1 ... Yk) is a dimension, then we will say that they are dimensions of X. Thus the *structure* of X is the set of all dimensions of X. Every dimension '(r X Y)' in CK-LOG will have a *converse*, '(cr Y X)', such that for instances x of X, and y of Y, [(r x y) IFF (cr y x)] is true in every world state. The pair (r, cr) is a *converse relation name pair*.

A dimension, '(r X Y)', is interpreted in CK-LOG as specifying that there are world states in the universe in which there exist instances x of the concept X, and y of the concept Y, for which '(r x y)' is true:

((r X Y) IFF
[(EXISTS U_t WORLD-STATE)(EXISTS x X)(EXISTS y Y)
(ISTRUE (r x y) U_t)])†

Thus, if '(r X Y)' has not been declared as a dimension then in every world state of the

† This is easily generalised to tuples '(X Y1 ... Yk).' The form '((EXISTS x X) exp)' stands for '((EXISTS x)(x ∈ X) AND exp))', and form '((EVERY x X) exp)' stands for '((EVERY x)(x ∈ X) IMPLIES exp))'. The general form of a quantifier is '((EVERY/EXISTS x range) exp)' where range is a set.

universe '(r x y)' will be false, for all instances x of X and y of Y. If X is a specialization of Z and '(r Z W)' is a dimension then Y should be a specialization of W**. The dimension '(r Z W)' is then said to be a generalization of the dimension '(r X Y)'.

If '(r X Y)' is a dimension then we shall say that '(r x y)' is an instance of '(r X Y)' if x is an instance of X, and y is an instance of Y. Clearly, not every instance of a dimension will be true in a world state. For a given instance x of X and given relation name, r, the ***logical restrictions*** associated with '(r X Y)' will specify the conditions under which '(r x y)' may be true in a world state, for a particular instance y of Y.

Thus, one may specify for (MIL-ACTION x y r)

```
[3.4].  ((EVERY x FORCE)(EVERY y FORCE)(EVERY r REGION)
         [(MIL-ACTION x y r)  IFF
          ([EXISTS mil-action MIL-ACTION]
           ([prosecuted-by mil-action x] AND
            [is-against mil-action y] AND
            [region-of mil-action r] AND [enemy-of x y]))])
```

to indicate that x is the prosecuting force, y the enemy force, and r is the region of the action. The above restriction will be associated with the anchor '(*instance-of* MIL-ACTION)' in the knowledge base. Similarly, the dimension '(specified-by MIL-ACTION OPORDER-TASK)' may have the restriction,

```
[3.5].  ([specified-by mil-action oporder-task] IMPLIES
             [specifies-tasks (ordered-by mil-action) oporder-task])].
```

i.e., the oporder-task that specifies a mil-action should be one of the tasks specified by the oporder that orders the mil-action. The term '(ordered-by mil-action)' in the above expression refers to this oporder. This restriction will be associated with the term '(specified-by MIL-ACTION)' in the knowledge base. Every dimension '(r X Y)' may thus have a logical restriction. The restriction on '(r X Y)' will be associated in the knowledge base with the term (called an *anchor*†) '(r X).' The logical restrictions at anchors, (r X), are written in

** Exceptions to this general rule are possible. Such exceptions are allowed only if certain designated flags are associated with the generalization '(r Z W)'.

† The create, destroy and keep actions associated with the dimension are also kept anchored at '(r X)' in the knowledge base. Besides these MDS uses '(r X)' as the key to a variety of other information associated with the dimensions (r X Y_j), for j = 1, 2, ..., n. Hence the name *anchor*.

CK-LOG in a special form. They are called ***consistency conditions*** (CC's). The CC's are used by TMS to detect ***contradictions*** in world states. This is discussed in the next subsection.

3.4. Consistency Conditions and Contradictions in World States.

For a dimension '(r X Y)', the consistency condition (CC) anchored at (r X) may have one of three forms:

definitional:	((EVERY x X) [(r x y) IFF (exp x y)]). In this case the restriction anchored at (r X) is stated as, {(y Y) | (exp x y)}.
implicational:	((EVERY x X)[(r x y) IMPLIES (exp x y)]). In this case the restriction anchored at (r X) is stated as, {(y Y) | ([r x y] AND [exp x y])}.
default:	((EVERY x X)[(exp x y) IMPLIES (r x y)]). In this case the restriction anchored at (r X) is stated as, {(y Y) | ([r x y] OR [exp x y])}.

Here (exp x y) is an arbitrary logical expression in the domain language DL in which x and y occur free. Notice that exp *could not thus contain modal expressions.* They may however contain timed expressions. I have used the word '*default*' above to name the reverse implication. In the three valued logical system this CC form may be used to define default values for an anchor. If '(r x y)' is unknown in a world state for all y, then in the *default* CC the set {y | (exp x y)} may be interpreted as defining the default values for the anchor (r x).

Let us denote the general form of a CC at '(r X)' by,**

[3.6]. $CC[r\ X] = \{(y\ Y_1\ Y_2 \ldots Y_n) \mid (\mathbf{ccexp}_{[r\ X]}\ x\ y)\}$.

where x is an instance of X, and '$(\mathbf{ccexp}_{[r\ X]}\ x\ y)$' is either '(exp x y)', or '([r x y] AND [exp x y])', or '([r x y] OR [exp x y])'. For the formulation of the CC's given above the following is always true:

** If '(r X Y_1)', '(r X Y_2)', ..., '(r X Y_n)' are all dimensions of X then the set former in the CC definition at the anchor '(r X)' will have the form shown here.

[3.7]. $[(\mathrm{r\ x\ y})\ \text{IFF}\ (\mathbf{ccexp}_{[r\ X]}\ \mathrm{x\ y})]$.

For the anchor (r x) in a world state $\mathbf{U}_t$, where x is an instance of X, the CC defined at (r X) is used by the truth maintenance system, TMS, to compute*,

[3.8].
$$\{\mathrm{y} \mid (\mathrm{ISTRUE}\ (\mathbf{ccexp}_{[r\ X]}\ \mathrm{x\ y}))\} = \mathbf{TP}_{CC[r\ x]}\ .$$
$$\{\mathrm{y} \mid (\mathrm{ISUNKNOWN}\ (\mathbf{ccexp}_{[r\ X]}\ \mathrm{x\ y}))\} = \mathbf{?P}_{CC[r\ x]}\ .$$
$$\{\mathrm{y} \mid (\mathrm{ISTRUE}\ (\mathbf{ccexp}_{[r\ X]}\ \mathrm{x\ y}))\} = \mathbf{FP}_{CC[r\ x]}$$

where **TP**, **?P** and **FP** stand for true, unknown and false parts of the partition induced by CC[r X] at (r x). Let

[3.9]. $\mathbf{P}_{CC[r\ x]} = [\mathbf{TP}_{CC[r\ x]}\ \mathbf{?P}_{CC[r\ x]}\ \mathbf{FP}_{CC[r\ x]}]$,

where $\mathbf{P}_{CC[r\ x]}$ is the *partition* induced at (r x) by CC[r X]. In general, if Z1, Z2, ..., Zk are all the generalizations of X, where Z1 = X, for which CC[r Zi] exist for $1 \leq i \leq k$, then the inheritance of the consistency conditions is governed by the following modification** to the definition [3.8] given above. Let $\mathbf{ccexp}_{[r\ Zi]}$ be the cc-expression in CC[r Zi]. Then $(\mathbf{ccexp}_{[r\ X]}\ \mathrm{x\ y})$ used in [3.8] above is changed to,

[3.10]. $(\mathbf{ccexp}_{[r\ X]}\ \mathrm{x\ y}) = [(\mathbf{ccexp}_{[r\ Z1]}\ \mathrm{x\ y})\ \text{AND} \ldots \text{AND}\ (\mathbf{ccexp}_{[r\ Zk]}\ \mathrm{x\ y})]$

and the partition $\mathbf{P}_{CC[r\ x]}$ is computed with this new $\mathbf{ccexp}_{[r\ X]}$†. This partition is compared with the partition,

[3.11]. $\mathbf{P}_{(r\ x)} = [\mathbf{TP}_{(r\ x)}\ \mathbf{?P}_{(r\ x)}\ \mathbf{FP}_{(r\ x)}]$,

at (r x) in the world state to detect contradictions. One may note that,

---

* If the second argument is missing in ISTRUE, ISFALSE, ISUNKNOWN then its default value is the current world state.

** Unless otherwise specified by appropriate use of exceptions.

† In MDS one may associate CC's also with anchors (r c) where c is a constant. There are situations where this is necessary to capture exceptions that apply only to particular constants of a concept X. If there is a CC associated with (r c), then in the above case the ccexp associated with (r c) will also appear as one of the conjuncts in [3.10].

[3.12a]. ($y \in \mathbf{TP}_{CC[r\,x]}$) IFF (r x y) is true as per the CC.
($y \in\ ?\mathbf{P}_{CC[r\,x]}$) IFF (r x y) is unknown as per the CC.
($y \in \mathbf{FP}_{CC[r\,x]}$) IFF (r x y) is false as per the CC.

Let 'CC[r x y]' denote this truth value induced by the CC on (r x y). Similarly,

[3.12b]. ($y \in \mathbf{TP}_{(r\,x)}$) IFF (r x y) is true in the world state.
($y \in\ ?\mathbf{P}_{(r\,x)}$) IFF (r x y) is unknown in the world state.
($y \in \mathbf{FP}_{(r\,x)}$) IFF (r x y) is false in the world state.

Let '(r x y)' by itself denote its truth value in the world state.

Then,

[3.13]. *Conditions for Contradiction Detection*: If for some y in $\mathbf{TP}_{CC[r\,x]}$ or $\mathbf{TP}_{(r\,x)}$, ((r x y) AND CC[r x y]) = F (false), then a contradiction is declared. Or, if for some Y_j such that (r X Y_j) is a dimension, the number of instances of Y_j in $\mathbf{TP}_{CC[r\,x]}$ is greater than the *upper bound* associated with the dimension then a contradiction is declared. This is called the *overflow* condition**.

If for no y in $\mathbf{TP}_{CC[r\,x]}$ or $\mathbf{TP}_{(r\,x)}$, ((r x y) AND CC[r x y]) = F and for some z in $\mathbf{TP}_{(r\,x)}$, ((r x z) AND CC[r x z]) = ? then a conditional acceptance of changes at (r x) is indicated.

If for all y in $\mathbf{TP}_{(r\,x)}$, ((r x y) AND CC[r x y]) = T then the changes at (r x) are unconditionally accepted.

In certain cases, as indicated by the 'update' flags associated with the dimensions, TMS may update the truth value of (r x y) in the world state to CC[r x y]. TMS has special facilities for keeping track of interactions between the CC's. Thus if the value of an anchor (r x) is changed, it can identify the set of all anchors (r′ z) whose logical restrictions are likely to be affected by the change, $\delta_{[r\,x]}$, at (r x). This is called the

** The overflow condition here is a difficult one to interpret with in the theorem proving context. Normally, when a contradiction is encountered TMS will return a logical expression (as discussed later in this section) that explains the contradiction. This expression may be used to set up new goals, if so desired, to remove the contradiction. When the contradiction is due to an overflow TMS cannot return any explanatory logical condition for this overflow. It will simply reject the changes, with a message to the user that a overflow had occurred.

dependency set of (r x) denoted by $\mathbf{D}_{[r\,x]}$. By analyzing the CC's defined at the various anchors, for each anchor (r X), MDS can construct logical expressions, called **filters**, such that for a given (r x),*

$$[3.14]. \quad \mathbf{D}_{[r'\,r\,x]} = \{\text{'}(r'\ z)\text{'} \mid (\mathbf{filter}_{[r'\,r\,X]}\ x\ z\ \delta_{[r\,x]})\}, \text{ and}$$
$$\mathbf{D}_{[r\,x]} = \text{UNION } \{\mathbf{D}_{[r'\,r\,x]}\} \text{ over } r'.$$

In this case to check for possible contradictions, TMS will check the CC's at every (r′ z) in $\mathbf{D}_{[r\,x]}$. It will accept the change at (r x) only if no contradiction is detected at any of the (r′ z). It will accept the change at (r x) *conditionally* if at one or more (r′ z) the associated ccexp evaluates to ? (unknown) and no contradictions are detected. If a contradiction is detected then the changes at (r x) will be rejected. TMS has special facilities for efficient evaluations of filters and for rechecking of CC's by saving partial results of previous evaluations.

If the changes at (r x) are unconditionally accepted then TMS will return a logical expression, $\mathbf{J}_{[r\,x]}$, for which (ISTRUE $\mathbf{J}_{[r\,x]}$ U_t) = T. This expression is interpreted as the justification for the acceptance of the changes at (r x): $\mathbf{J}_{[r\,x]}$ is the strongest condition such that the changes at (r x) logically imply $\mathbf{J}_{[r\,x]}$ in $\mathbf{U}_t$.

If the changes at (r x) are conditionally accepted then TMS will return an expression, $\mathbf{C}_{[r\,x]}$, such that (ISUNKNOWN $\mathbf{C}_{[r\,x]}$ U_t) = T, and the changes at (r x) logically imply $\mathbf{C}_{[r\,x]}$. $\mathbf{C}_{[r\,x]}$ is interpreted as expressing the weakest condition under which the changes will be acceptable. It is used in a problem solving process to generate the ***hypotheses*** associated with the solution of a problem (as discussed in section 2).

If the changes at (r x) are rejected then TMS will return the expression $\mathbf{N}_{[r\,x]}$†, such that (ISFALSE $\mathbf{N}_{[r\,x]}$ U_t) = T, and the changes at (r x) logically imply $\mathbf{N}_{[r\,x]}$. $\mathbf{N}_{[r\,x]}$ is interpreted as expressing the weakest condition which should be made true in order for the changes at (r x) to be acceptable. It may be used in a problem solving process to set up the new subgoals,

* This subsystem has not yet been implemented. At present filters are defined by users.

† $\mathbf{N}_{[r\,x]}$ will not contain expressions for indicating overflows.

[3.15]. → <conjunction of changes made in the world state> ;
→ (ACHIEVE N) ;

under user's option. The successful solution of these two subgoals, will force the changes at (r x) to be accepted. The expressions returned by TMS are called *residues*. The calculus for residue extraction and properties of residues are discussed in [Srinivasan 1973, '77]**.

The ASSERT statement is used in CK-LOG to incorporate changes into world states. The ASSERT statement received by TMS will in general have a list of grounded (timed) literals as arguments. All the literals in the statement are asserted in parallel, in the sense that the changes specified by the assertion are first incorporated into the world state, and the contradiction check is performed afterwards on the new world state so obtained. The form '(ASSERT (*instance-of* X c))' is interpreted as a command to create in the world state the constant c as an instance of the concept X, and '(ASSERT (NOT (*instance-of* X c)))' is interpreted as a command to destroy the constant c. Special procedures CRI (CReate Instance), and DESI (DEStroy Instance) are used in TMS to implement these.

Every object, concept and action, c, in the world state, and every property associated with c (via a relation) will have a creation (and a destruction) time associated with it. This time will be specified by the time expression appearing in the asserted timed literal. If the literal asserted is not a timed literal then the *current time* in the world state is associated with it. It is of course true that c can have a property at time, t, only if it had been itself created at t or before t.

(t (OCCURS c)) = (OCCURS (t c)) = (OCCURS (t (*instance-of* c X))), where X is the concept of c, is true if c occurs (exists) in the world state U_t *. For a closed proposition not containing modal expressions, (ISTRUE (t proposition)) is true if the propositition is true in U_t . In this case (OCCURS (t proposition)) = (ISTRUE (t proposition)).† Similar

** For examples of residues and their uses see [Sridharan 1978], [Srinivasan 1973, '77, '81].

* In general, a constant c can be simultaneously an instance of several concepts, and so also a concept X may be a specialization of several other concepts. If c is an instance of both concepts X and Y then the properties associated with c will be defined by the union of the dimensions associated with X and Y. The logical restriction associated with each dimension of c will be the conjunction of the restrictions associated with the corresponding dimensions in X and Y. If a concept X is a specialization of both Y and Z then X will inherit the dimensions of both X and Y. The properties inherited by X from Y and Z may be controlled by the CC's associated with X and by the use of designated inheritance flags.

† It may be noted that (ISTRUE (CREATE exp)) = (ISTRUE exp)), and (OCCURS (ASSERT exp)) = (OCCURS

considerations apply to ISFALSE and ISUNKNOWN operators. The inference rules for these were discussed in section 2.9.

For modal expressions that refer to processes like CREATE, DESTROY, ACHIEVE, etc., (OCCURS (t process-exp)) ═ (t (OCCURS process-exp)) will be true in U_t only if all the actions that are implicitly referred to by the process-exp occur in the world state. It will be false in U_t if some or all of the actions do not occur in the world state. It should be noted that the truth of (OCCURS (t process-exp)) does not depend on the success or failure of these actions. Thus (OCCURS (t (CREATE **exp**))) will be true if all the actions needed to make **exp** true exist in U_t.

As discussed in section 2, CK-LOG uses its inference engine to analyze process expressions and reduce them to the actions implicitly denoted by them. In describing the inference rules used for this analysis I introduced the meta-functions, **create-action-of**, **destroy-action-of** and **keep-action-of** to retrieve the actions associated with a positive literal. As mentioned before, these action definitions are associated in the knowledge base with anchors. The form of these definitions is discussed in the next subsection.

3.5. Actions Associated with Anchors.

The create, destroy and keep actions that are associated with anchors (r X) are stated as inference rules. The actions that might be involved in the creation, destruction or the maintenance of an instance, c of a concept X, are associated with the '(*instance-of* X)' anchor. The definitions of these actions is best introduced through a few examples.† The first example in table 3-I shows the *destroy-action* for destroying the truth of '(belongs-to BASE FORCE)', i.e. for destroying the fact that a base might belong to a given force. As mentioned in section 2, the invocation of this action will be caused by a '(DESTROY (belongs-to base force))' statement encountered in a problem solving process, in a world state in which the said base does belong to the force. The condition associated with the rule in the table says that it could be invoked unconditionally (reader may contrast this with the *create-action* shown in table 3-II, which has an invocation condition

exp).

† These examples are again taken from the OPPLAN-CONSULTANT. See [Srinivasan 1984] for details on how the rules shown here are used in a planning problem solving context.

associated with it). The rule shown in table 3-I is anchored to '(belongs-to BASE)', as its *destroy-action*.

TABLE 3-I: *destroy-action* for (belongs-to BASE FORCE).

Condition	Inference
T	((EXISTS x FORCE) ([NOT (controls (region-of base) x) AND [CREATE (tm-term (controls (region-of base) x)]])) (tm-term (belongs-to base force))

When this rule is invoked by the **destroy-action-of** function it will do appropriate substitutions for the variables base and force, and the pattern 'tm-term', by matching the expression at the bottom of the inference rule in table 3-I with the argument of the DESTROY expression that caused the invocation. During this process the function will also substitute all the terms appearing in the top expression of the inference rule with their values, if such values exist in the world state at the time of the invocation. The term '(region-of base)' appears in the top expression in table 3-I. This is the region where the base is located. A part of the general knowledge of the system (for the operational planning domain), as expressed by the CC associated with the anchor, '(belongs-to BASE)', is that a base belongs to the force that controls its region. Thus, to destroy this belongs-to relation the rule in table 3-I says that one should find a FORCE, x, who does not control the region of the base and this force x should then be made to take control of this region. The point to note here is that these action rules are stated in TML and thus modal expressions may occur in the statement of these rules.

The rule in table 3-II specifies the *create-action* for '(bases-at REGION BASE). This rule will be invoked, of course, only if the base is not in the given region at the time it is invoked. The rule says that to create a base in a region at a time specified by the 'tm-term', a BUILD-BASE action, x, should be created by the agent who controls the region of the base, the base-of (the base to be built by) this action should be the base, its region should be region, and its starting-time should be the needed time (needed-time-for) before the ending-time specified by the tm-term. The function MINUS-TIME is one of the functions defined in MDS. It takes as arguments time instants or time expressions and returns either time instants or reduced time expressions. The condition on this rule says that

TABLE 3-II: *create-action* for (bases-at REGION BASE).

Condition	Inference
[NOT (OCCURS base))]	[(EXISTS x BUILD-BASE) ((agent-of x (controls region)) AND (base-of x base) AND (region-of x region) AND (starting-time-of x (before (MINUS-TIME tm-term (needed-time-for x)))) AND (ending-time-of x tm-term))]
	(tm-term (bases-at region base))

the rule may be invoked only if base does not already exist in the world state (in any region).

TABLE 3-III: *Create-action* for (location-of OBJECT LOCATION).

Condn.	Inference
T	((EXISTS trans TRANSPORT)(EXISTS convoy CONVOY) (EXISTS x LOCATION) ((before tm-term)(location-of object x)) AND (CREATE ((starting-time-of trans)(location-of convoy x))) AND (belongs-to convoy (controls (region-of location))) AND (objects-of trans object) AND (destination-of trans location) AND (ending-time-of trans tm-term) AND etc.)
	(location-of object location)

The rule in table 3-III specifies the action for making an instance of '(location-of OBJECT LOCATION)' true in a world state, i.e. to put an object at a location. This rule will be invoked, of course, only if the object is not already in the given location. The rule in effect calls for the formation of a convoy to transport the object (using trans) from its current location to its new location, with the appropriate timing conditions.

Rules like these are invoked only by the problem solving system of CK-LOG. *They are not invoked by* TMS *during its model building (updating) processes.* The action definitions illustrated above enables CK-LOG to identify the actions that are implicitly referenced by the process expressions. It also provides the facility to build the *possible worlds* associated with the *modal expressions*. The organization presented here provides a general logical scheme to describe and reason about actions, taking into consideration their time dependencies. The crucial facility in CK-LOG that makes this possible is the organization of its TPS, in which the TPS uses and updates models of world states during the theorem proving process. The states of the models so built restrict the inferences that are made by TPS. This makes it possible to use CK-LOG to plan for the achievement of given objectives, as discussed in [Srinivasan 1984] and in section 4.

In examples shown above the conditions associated with the rule invocation are rather simple ones. As experience develops with the use of CK-LOG in a particular domain one may find it profitable to define different specializations of rules of this kind, each associated with a different world state condition for its invocation. Thus with experience one may find an expanding repertoir of inference rules in the system with different invocation conditions associated with them. One may view these invocation conditions as expressing the heuristics for rule selection. These conditions are not used to modify the sequents in a deduction tree, they are used only for rule selection.

This will make it possible in CK-LOG to learn through problem solving experience. As I mentioned before, since CK-LOG's inference engine can run both ways (i.e. for deductive as well as inductive inference), the inference rules may be used to generalize the world state situations under which certain patterns of rule applications succeed in a proof tree. These generalizations may be associated as invocation conditions with possibly new versions of the invoked rules. Each such new rule will be a specialization of the original rule. At a later time if this specialized rule fails after invocation, then the negation of the conditions causing failure may be used to strengthen the invocation condition associated with this rule. This scheme provides the basic ingredients necessary for learning through experience. It may be noted that the *theory forming ability*, namely the ability to generalize given situations in a world state, is essential to this kind of learning. It is now too early to comment on the use of this scheme in CK-LOG. Little is known about its theory and practice. But it is significant that as a logic based knowledge processing system CK-LOG has this dual potential, (a). for knowledge based problem solving, and (b). for knowledge based learning. This is unique to CK-LOG organization.

Let me now present an example that illustrates the use of CK-LOG's action calculus for planning, analysis and execution of actions.

4. The Action Calculus of CK-LOG, An Illustration.

I present here a relatively simple situation consisting of the action GIVE and BUY, and show how the action calculus implied by the inference rules for the modal operators may be used to solve problems that involve action execution, planning and analysis of action consequences.

Figure 4.1: Dimensions common to all ACTIONs.

structure	(starting-time-of ACTION TIME), 1 (ending-time-of ACTION TIME), 1 (needed-time-for ACTION DURATION), 1 (status-of ACTION ACTION-STATUSs), 1

Figure 4.1 shows some of the dimensions common to all actions, the ones we will be using in this example. The GIVE concept is shown in figure 4.2 The tuple definition that occurs first in the *structure* of GIVE indicates the conditions under which two given actions at a given time are to be reckoned as being distinct. Every GIVE action has an agent, a recipient and items that are given. Its *function* says that after the interval of a give there is an object whose ownership changes, and/or there is a service which the recipient of the give receives from the agent of give. Its *behavior* states that every item that is given should be owned by the giver before the interval of give and for every service that is given, the giver should be able to perform it. Also, during the interval of give the recipient either enjoys it or suffers it. The CREATE operators in these expressions indicate the properties that are newly created.

Let me state the CC associated with (status-of GIVE) just for the case of successful completion of the action:

[CC]: {x | [(status-of give x) OR
((is x successful) AND
(owns (recipient-of give) (items-of give)))]}

The 'is' relation is used in CC's to bind the set variable as shown above**. This is a *default* CC. If the status is unknown and the recipient of 'give' owns the items-of 'give' then the status will be by default set to successful.

** This relation is used in order to be able to write all CC's in the same standard format. It is suppressed, and appropriate substitutions for the set variable are made, when ccexps are transferred to problem sequents in the KB-lookup process discussed in section 2.

Figure 4.2: The GIVE concept.

structure	(GIVE (agent-of give)(recipient-of give)(items-of give)) (agent-of GIVE PERSON), 1 (recipient-of GIVE PERSON), 1 (items-of GIVE (OBJECTs SERVICEs)), 1
function	[(before (ending-time-of give)) ([(EXISTS x OBJECT) ([items-of give x] IMPLIES [CREATE ([owns (recipient-of give) x] AND [NOT (owns (agent-of give) x)])])] AND [(EXISTS x SERVICE) ([items-of give x] IMPLIES [(agent-of x (agent-of give)) AND (CREATE [recipient-of x (recipient-of give)])])])]
behavior	([(EVERY x OBJECT) ([items-of give x) IMPLIES [CREATE ((before (starting-time-of give))(owns (agent-of give) x))])] AND [(EVERY x SERVICE) ([items-of give x] IMPLIES [CREATE ([(before (starting-time-of give))(can-perform (agent-of give) x)] AND [(during (*interval-of* give)) ([enjoys (recipient-of give) x] OR [suffers (recipient-of give) x])])])])

The concept of BUY is shown in figure 4.3. This has only *structure* and *behavior* defined for it. Every time a BUY occurs this behavior will occur in the world state. The behavior simply says that the seller gives the buyer the items bought and the buyer gives the seller the cost of the items.

The structures for SELLER and COST are shown in figure 4.4. The *create-action* for '(owns PERSON OBJECT)' is shown in table 4.I. It says that to own something a person should find a seller who sells it and buy it from him at the appropriate time. I have indicated above that the CREATE expression should become true before the time specified by the tm-term. Normally in action statements like this one would specify that the necessary actions should start the needed time before the tm-term. The needed time itself will be specified in the action definition as a CC (See [Srinivasan 1984] for an example of this kind). I have chosen the simpler course above.

Figure 4.3: The BUY concept.

structure	(BUY (buyer-of buy)) (seller-of BUY SELLER), 1 (buyer-of BUY PERSON), 1 (items-of BUY (OBJECTs SERVICEs)), 1 (status-of BUY ACTION-STATUSs)
behavior	[(between (*interval-of* buy)) ([EVERY x (items-of buy)][EXISTS y (cost-of (seller-of buy))] [(item-of y x) IMPLIES (CREATE [(GIVE (seller-of buy)(buyer-of buy) x) AND (GIVE (buyer-of buy)(seller-of buy) (price-of y))])])]

Figure 4.4: Structures of SELLER and COST.

structure	(sells SELLER (OBJECTs SERVICEs)), 1, (cost-of SELLER COSTs), 1,
structure	(item-of COST (OBJECT SERVICE)), 1, (price-of COST DOLLARs),

TABLE 4.I: *Create-action* for (owns PERSON OBJECT).

Condn.	Inference
T	([EXISTS seller SELLER] [(sells seller object) AND [CREATE ([before tm-term] [(EXISTS buy BUY) ([seller-of buy seller] AND (buyer-of buy person) AND (items-of buy object)])])])
	(tm-term (owns person object))

Let us now consider the problem to make true the statement that John owns a car after [1984 Sept]:

[P1]. → (CREATE [(EXISTS x CAR)([after (1984 Sept)] [owns John x])]) ;

Let us assume that John is already in the world state associated with [P1], and there are numerous instances of SELLERs who sell cars, where the cars being sold are also in the world state. Let us assume that cars are created by simply asserting into the world state the appropriate properties of the cars, whatever they might be*, and that the current time in the world state is [1984 Sept 27]. The sequence of deductions on this problem is shown in tables 4.II through 4.VIII. In these tables the rules in the last column of the table are applied to the [*]ed expressions in the sequent in the center column. The number, if any, appearing in parenthesis after the rule indicates the number of times the rule is applied.

The [→ CRR-E] rule is applied first resulting in the problem [P2]. The CREATE expression is still retained in the sequent. I have used CREATE(1) in [P2] to indicate that this CREATE expression has been already expanded once. At this point the candidate constants for binding eGx1 are the constants that exist in the world state. There are no *create-action* associated with (*instance-of* CAR). So the application of [CRR-EL2] rule to '(CREATE (eGx1 ∈ CAR))' simply transforms it to an ASSERT statement as shown in [P3]. The second create expression in [P2] gets transformed to the *create-action* associated with the ownership relation, shown in table 4.I, with proper variable and tm-term substitutions. Notice that '(before (after [1984 Sept]))' got transformed in this process to '(before [1984 Oct 1])'. This happens as a part of the evaluation of terms that occurs during the invocation of actions using **create-action-of** function in [CRR-EL2]. The resultant sequent is shown in [P3].

At this point the [CRR-EL2] rule is applied 4 times to the CREATE expressions in [P4] and then the assert elimination rule, [→ ASRT-EL], is applied. In this case there are no *create-actions* associated with the atoms appearing in the CREATE expressions in [P4]. This causes the **creation-action-of** function to replace 'CREATE' with 'ASSERT' in the expressions in [P4]. Subsequent application of [ASRT-ASRT] rule removes the nested asserts, and finally [→ ASRT-EL] changes the ASSERT to **R-assert**. The resultant sequent is shown in [P5].

Now, we have a simple parallel assertion on the right side with variables eGx1, eGx2 and eGx3. All these variables should eventually be bound to the constants that

* We will not get involved with the car creation processes.

TABLE 4.II: Deductions on P1.

Name	Sequent(s)	Rule(s)
[P1]	→ (CREATE ((EXISTS x CAR)...))[*] ;	[→ CRR-E]
[P2]	→ (ASSERT ([CREATE (eGx1 ∈ CAR)],[*] [CREATE ([after (1984 Sept)][owns John eGx1])])),[*] (CREATE(1) ((EXISTS x CAR)...)) ;	[CRR-EL2](2)
[P3]	→ (ASSERT ((ASSERT (eGx1 ∈ CAR)),[*] ([EXISTS seller SELLER] [(sells seller eGx1) AND (CREATE ([before (1984 Oct 1)] [(EXISTS buy BUY) ([seller-of buy seller] AND [buyer-of buy John] AND [items-of buy eGx1])])])])))),[*] (CREATE(1) ((EXISTS x CAR)...)) ;	[→ ASRT-E] [→ CRR-E] [TMR-AND] [CRR-AND] [ASRT-AND] [ASRT-ASRT]
[P4]	→ (ASSERT ((eGx1 ∈ CAR),(eGx2 ∈ SELLER),(sells eGx2 eGx1), (CREATE ([before (1984 Oct 1)][eGx3 ∈ BUY])), (CREATE ([before (1984 Oct 1)][seller-of eGx3 eGx2])), (CREATE ([before (1984 Oct 1)][buyer-of eGx3 John])), (CREATE ([before (1984 Oct 1)][items-of eGx3 eGx1])))),[*] (CREATE(1) ((EXISTS x CAR)...)) ;	[CRR-EL2](4) [→ ASRT-EL]
[P5]	→ (**R-assert** ((eGx1 ∈ CAR),(eGx2 ∈ SELLER),(sells eGx2 eGx1), ([before (1984 Oct 1)][eGx3 ∈ BUY]), ([before (1984 Oct 1)][seller-of eGx3 eGx2]), ([before (1984 Oct 1)][buyer-of eGx3 John]), ([before (1984 Oct 1)][items-of eGx3 eGx1])),[*] (CREATE(1) ((EXISTS x CAR)...)) ;	
[P6]	(**value-of** (*behavior* buy-3)) → (CREATE(1) ((EXISTS x CAR)...)) ;	

exist in the world state associated with sequent [P4]. The **R-assert** function will now seek to bind these variables in the appropriate manner, invoking the help of the user whenever there are choices to be made. None of the atoms in the assert have matching atoms on the left side. The conjunction of these atoms is thus presented to the user as a possible hypothesis:

[HP4]. → ((eGx1 ∈ CAR) AND (eGx2 ∈ SELLER) AND (sells eGx2 eGx1) AND
([before (1984 Oct 1)][eGx3 ∈ BUY]) AND
([before (1984 Oct 1)][seller-of eGx3 eGx2]) AND
([before (1984 Oct 1)][buyer-of eGx3 John]) AND
([before (1984 Oct 1)][items-of eGx3 eGx1])) ;

HYPOTHESIS? >

Let us suppose that the user responds with 'Y', confirming the hypothesis. This hypothesis is now associated with [P4] and also entered as a new problem in the frontier set of the deduction tree†. The assertion is now performed. The assimilation of the assertion occurs in the following order: The instantiation commands in the assertion are first executed. In the above case there are no such commands (if any, they will be associated with the instantiation variables in an assertion). After this the generalization variables in the assertion are checked to see whether they have candidate bindings in the world state. In the above case all except eGx3 have candidate bindings (there are no active instances of BUY in the world state). This causes the system to create a new instance of BUY, say buy-3 and bind it to eGx3. Substituting buy-3 for eGx3 in the assertion produces the grounded atom, '(buyer-of buy-3 John),' which is now asserted into the world state. Also, the starting time of buy-3 is set to '(before [1984 Oct 1])', since this is the time it was created as per the assertion above. I have not specified the needed time for a BUY action. Let me assume that it is negligible, and that the ending-time-of buy-3 is also '(before [1984 Oct 1])'.

At this point the world state is searched to see whether there are any possible bindings for the other generalization variables that occur in the assertion, selecting the variables in the order of their creation. This is done by evaluating the set expressions given below over the world state:

[S1]. {(eGx1 CAR) | [(sells eGx2 eGx1) AND ([before (1984 Oct 1)][items-of buy-3 eGx1])]}
[S2]. {(eGx2 SELLER) | [(sells eGx2 eGx1) AND ([before (1984 Oct 1)](seller-of buy-3 eGx2))]}

These set expressions are formed using the atoms that contain the set variables in the given assertion. The logical conditions that define these will now evaluate to the truth value, ?, in the world state, because the relevant properties of buy-3 are undefined. Thus

† I have not shown this hypothesis problem in the tables shown here.

the bindings generated for these variables are interpreted as candidate bindings for the variables. These candidate bindings are now presented to the user requesting the desired selections to be made. Suppose the selections eGx1 = car-1 and eGx2 = seller-2 are made by the user. These are then used to recheck the satisfaction of the conditions in the assertion. If seller-2 does not sell car-1 then the candidate dealers who sell car-1, and the candidate cars that are sold by seller-2 will both be presented to the user. Once the appropriate choices are made all the three variables are bound to their respective selected values and the world state is appropriately updated**.

In the simple case above the assertion succeeds unconditionally. The new instantiation of BUY now results in updating [P5] with the action predicate for buy-3 producing the new problem sequent shown in [P6]. The action predicate here is simple because buy-3 has no *function* associated with it. The ASSERT statement itself does not appear in [P6] since it has been already successfully accomplished. The evaluation of the **value-of** function now results in sequent [P7] shown in table 4.III. Notice that during this evaluation all the terms in (*behavior* buy-3) are evaluated in the world state and their values are substituted in the expression. The deductions from [P7] are shown in table 4.III.

Once [P10] is obtained the create elimination rule, [CREL-2], is applied, since the GIVE expressions are not true in the existing world state. Since there is no *create-action* associated with GIVE this causes the **create-action-of** function to simply replace 'CREATE' by 'ASSERT' in [P10]. Subsequent application of [ASRT-ASRT] rule removes the nested ASSERTs from the resultant expression, producing [P11]. At this point the axiom test is applied to bind the variables uGx1 and elc1. The interpretation associated with $\in$ causes uGx1 to be bound to car-1. The range for elc1 is the set of costs associated with seller-2. The cost in the world state that can now match with the atom, '(item-of elc1 uGx1)', on the right side will thus be the COST that specifies the price-of car-1. Let us call this COST, cost. Let us suppose that the price of car-1 is 6000 dollars. Then the following will be true in the world state:

[(item-of cost car-1),(price-of cost (6000 DOLLARS))]

** Instead of making the selection of the car arbitrary, I could have given a description of the kind of car that John wants to own. I took the simpler approach here to keep the discussion short.

TABLE 4.III: Deductions on [P6].

Name	Sequents	Rules
[P7]	[(between (*interval-of* buy-3)) ([EVERY x {car-1}][EXISTS y (cost-of seller-2)] [(item-of y x) IMPLIES (CREATE [(GIVE seller-2 John x) AND (GIVE John seller-2 (price-of y))])])])] → (CREATE(1) ((EXISTS x CAR)...)) ;	[→ U]
[P8]	[(between (*interval-of* buy-3)) ((EXISTS y (cost-of seller-2)) [(item-of y uGx1) IMPLIES (CREATE [(GIVE seller-2 John uGx1) AND (GIVE John seller-2 (price-of y))])])] → (uGx1 ∈ {car-1}), (CREATE(1) ((EXISTS x CAR)...)) ;	[E →] [IMP →]
[P9]	(elc1 ∈ (cost-of seller-2), [(between (*interval-of* buy-3)) (CREATE [(GIVE seller-2 John uGx1) AND (GIVE John seller-2 (price-of elc1))])] → (uGx1 ∈ {car-1}), (item-of elc1 uGx1), (CREATE(1) ((EXISTS x CAR)...)) ;	[CRR-TM] [TMR-AND] [CRR-AND]
[P10]	(elc1 ∈ (cost-of seller-2), (ASSERT [(CREATE [(between (*interval-of* buy-3))(GIVE seller-2 John uGx1))]), (CREATE [(between (*interval-of* buy-3))(GIVE John seller-2 (price-of elc1))])]) → (uGx1 ∈ {car-1}), (item-of elc1 uGx1), (CREATE(1) ((EXISTS x CAR)...)) ;	[CREL-2] [CREL-2] [ASRT-ASRT]
[P11]	.., (ASSERT [((between (*interval-of* buy-3))(GIVE seller-2 John uGx1)), ((between (*interval-of* buy-3))(GIVE John seller-2 (price-of elc1)))]) → ..., (CREATE(1) ((EXISTS x CAR)...)) ;	[ASRT-EL →]
[P12]	.., (**L-assert** [((between (*interval-of* buy-3))(GIVE seller-2 John car-1)), ((between (*interval-of* buy-3))(GIVE John seller-2 (price-of cost)))]) → ..., (CREATE(1) ((EXISTS x CAR)...)) ;	

As a result of a successful match between '(item-of elc1 uGx1)' and '(item-of cost car-1),' elc1 now gets bound to cost. With these bindings, the application of [ASRT-EL →], changes the ASSERT expression in [P11] to [P12]*.

The **L-assert** function in [P12] now causes two new instances of GIVE to be created, with the indicated time conditions. Let us call these give1 and give2. These are shown below:

(GIVE seller-2 John car-1)	(GIVE John seller-2 (price-of cost)
(agent-of give1 seller-2)	(agent-of give2 John)
(recipient-of give1 John)	(recipient-of give2 seller-2)
(items-of give1 car-1)	(items-of give2 (6000 DOLLARS))

Both the starting and ending times for these give actions are between the (*interval-of* buy-3), and all the properties shown above are true during the interval between the starting time and the ending time of the GIVE actions. At this point the *function* and *behavior* of these actions are used to introduce the action predicates for these actions into sequent [P12] resulting in sequent [P13] shown in table 4.IV. The two GIVE actions above occurred within a parallel assert. Thus the *function* and *behavior* of both of these should be successfully completed before the assertion is considered to be successful. This is indicated in sequent [P13] by the *parallel predicate flag*, '$' associated with expressions in the sequent.

The rules [EVTM-EL3] and [IMP →] are applied to the expressions in [P13]. The [EVTM-EL3] rule is applied because the event conditions appearing in the expressions in [P13] are not true in the world state associated with the sequent. This rule transforms the causal expressions in [P13] to their equivalent implicational forms. Subsequent applications of [IMP →] rule to the resultant expressions produce the sequents [P14a] through [P14e]. Not all the sequents generated by the application of [IMP →] rule are shown in the table. Notice that, in the binding conditions for the time variables elt1 and elt3, the times when the status of the actions become successful have been set equal to the ending times of the respective actions. This is a piece of information that is known

* If John had bought several items then at this point, for each item bought by John, two GIVE assertions of this kind shown in [P12] would have been created. This will happen because uGx1 is a unversal generalization variable. Also, all the GIVE actions would occur in the same world state, since they are parallel actions. In this case, if John did not have enough money to pay for all the items he bought, then the goal would fail. Thus even though John might be able to buy x alone, or y alone, he might not be able to buy both of them.

for all actions: the ending time for an action is the same as the time when the action becomes successful. At the time the binding conditions are generated this is recorded. Also, when the binding conditions are generated all known partial orderings among the time points are recorded. This results in the orderings shown in the binding conditions. This ordering is now imposed on the expressions associated with the time instants. The expressions are selected for analysis in this order.

TABLE 4.IV: Deductions from [P13].

Name	Sequents	Rules
[P13]	$[(before (*time-of* (status-of give1 successful)))(**value-of** (*function* give1))][*], $[(before (*time-of* (**value-of** (*function* give1)))(**value-of** (*behavior* give1))][*], $[(before (*time-of* (status-of give2 successful)))(**value-of** (*function* give2))][*], $[(before (*time-of* (**value-of** (*function* give2)))(**value-of** (*behavior* give2))][*], → ..., (CREATE(1) ((EXISTS x CAR)...)) ;	[EVTM-EL3](4) [IMP →](4)
[P14a] [P14b] [P14c] [P14d] [P14e]	((before elt1)(**value-of** (*function* give1))),[*] ((before elt2)(**value-of** (*behavior* give1))),[*] ((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))) → ..., (CREATE(1) ((EXISTS x CAR)...)) ; .. → ..., (CREATE(1) ((EXISTS x CAR)...)),(elt1 (status-of give1 successful)) ; .. → ..., (CREATE(1) ((EXISTS x CAR)...)),(elt2 (**value-of** (*function* give1)) ; .. → ..., (CREATE(1) ((EXISTS x CAR)...)),(elt3 (status-of give2 successful)) ; .. → ..., (CREATE(1) ((EXISTS x CAR)...)),(elt3 (**value-of** (*function* give2)) ; **Binding conditions:** (elt1 = (*time-of* (status-of give1 successful)) = (ending-time-of give1)), (elt2 = (*time-of* (**value-of** (*function* give1))), (elt3 = (*time-of* (status-of give2 successful)) = (ending-time-of give2)), (elt4 = (*time-of* (**value-of** (*function* give2))), (before elt1 elt2), (before elt3 elt4)	

At this point let me consider the analysis of the two [*]ed expressions in sequent [P14a]. The analysis of the remaining expressions in this sequent will be similar to this. The expanded versions of these expressions are shown in sequent [P14a] in table 4.V.

Based on the time ordering of the expressions the [*]ed expression in [P14a] of table 4.V is now chosen for further analysis. The application of [TMR-AND] rule moves the time term inside the conjunction, distributing it to the two conjuncts. Subsequent application of [AND →] replaces the AND with a comma. At this point [TMR-U] rule is

TABLE 4.V: Expanded Sequent [P14a].

Name	Sequents	Rules
[P14a]	((before elt3)(value-of (*function* give2))), ((before elt4)(value-of (*behavior* give2))) ((before elt1) [(before (ending-time-of give1)) ([(EXISTS x OBJECT) ([items-of give1 x] IMPLIES [CREATE ([owns John x] AND [NOT(owns seller-2 x)])])] AND [(EXISTS x SERVICE) ([items-of give1 x] IMPLIES [(agent-of x seller-2)) AND (CREATE [recipient-of x John)])])])]),	[TMR-XN] [TMR-E] [AND →] [E →](2)
[*]	((before elt2) ([(EVERY x OBJECT) ([items-of give1 x) IMPLIES [CREATE ((before (starting-time-of give1))(owns seller-2 x))])] AND [(EVERY x SERVICE) ([items-of give1 x] IMPLIES [CREATE ([(before (starting-time-of give1))(can-perform seller-2 x)] AND [(during (*interval-of* give1)) ([enjoys John x] OR [suffers John x])])])])),[*] ... , → ..., (CREATE(1) ((EXISTS x CAR)...)) ;	[TMR-AND] [AND →] [U →](2)
[P14b]	.. → ..., (CREATE(1) ((EXISTS x CAR)...)),(elt1 (status-of give1 successful)) ;	
[P14c]	.. → ..., (CREATE(1) ((EXISTS x CAR)...)),(elt2 (value-of (*function* give1)) ;	

applied to move the time terms inside the respective universally quantified expressions. After this [U →] is applied twice, and there after [IMP →] is applied. This results in the sequents shown in [P15a] through [P15d] shown in table 4-VI.

At this point let me focus attention only on the <object-exp> shown in table 4.VI. The rule [TMR-IMP] is applied to the <object-exp> in sequents [P15a] and [P15c]. This causes the time condition to be moved in front of the antecedent and consequent of the implication. Subsequent application of [IMP →] results in the sequents [P16a] through [P16d] shown in table 4.VII.

TABLE 4.VI: Deductions from [P14a].

Name	Sequents	Rules
[P15a]	((**before** elt3)(**value-of** (*function* give2))), ((**before** elt4)(**value-of** (*behavior* give2))), [(**before** elt1)(*function* give1)], [(**before** elt2) ([items-of give1 uGx14) IMPLIES [IMP →] [CREATE ((before (starting-time-of give1))(owns seller-2 uGx14))])], [*I will call the above expression* <object-exp>] [(before elt2) ([items-of give1 uGx15] IMPLIES [CREATE ([(before (starting-time-of give1))(can-perform seller-2 uGx15)] AND [(during (*interval-of* give1)) ([enjoys John uGx15] OR [suffers John uGx15])])])])), ... , [*I will call the above expression* <service-exp>] → ..., (CREATE(1) ((EXISTS x CAR)...)) ;	[TMR-IMP]
[P15b]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)], <service-exp> → ..., (CREATE(1) ((EXISTS x CAR)...)),(uGx14 ∈ ((before elt2) OBJECT)	
[P15c]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)], <object-**exp**> → ..., (CREATE(1) ((EXISTS x CAR)...)),(uGx15 ∈ ((before elt2) SERVICE)	[TMR-IMP] [IMP →]
[P15d]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)], → ..., (CREATE(1) ((EXISTS x CAR)...)),(uGx15 ∈ ((before elt2) SERVICE), (uGx14 ∈ ((before elt2) OBJECT) ;	

The application of [CRR-TM] rule to the [*]ed expressions in [P16a] and [P16c] causes the '(before elt2)' term to be moved inside the CREATE expression. Subsequent application of the [TMR-XN] rule causes '(before elt2)' to be compared with '(before (starting-time-of give1))'. Elt2 here is the time when the *function* of give1 becomes true. The starting time of any action is always before the time when its *function* becomes

TABLE 4.VII: Deductions from [P15a] and [P15c].

Name	Sequents	Rules
[P16a]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)], ((before elt2) (CREATE ((before (starting-time-of give1))(owns seller-2 uGx14))))[*], <service-exp> → ..., (CREATE(1) ((EXISTS x CAR)...)) ;	[CRR-TM] [TMR-XN] [CREL-1]
[P16b]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)], <service-exp> → ...,(CREATE(1) ((EXISTS x CAR)...)),((before elt2)(items-of give1 uGx15))	
[P16c]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)], ((before elt2) (CREATE ((before (starting-time-of give1))(owns seller-2 uGx14))))[*], → ...,(CREATE(1) ((EXISTS x CAR)...)),(uGx15 ∈ ((before elt2) SERVICE) ;	[CRR-TM] [TMR-XN] [CREL-1]
[P16d]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)], → ...,(CREATE(1) ((EXISTS x CAR)...)),(uGx15 ∈ ((before elt2) SERVICE), ((before elt2)(items-of give1 uGx14)) • substitution: [uGx14 car-1]	•

true. Indeed, the time when the *function* of an action becomes true is always between the *interval-of* the action, if at all it becomes true. This again is a general piece of knowledge built into the time analysis functions. This causes the time term '(before elt2)' to be dropped by the **tmxn** function from the CREATE expression.

At this point, to apply any of the CREATE elimination rules it is necessary to test whether '(before (starting-time-of give)) (owns seller-2 uGx14))' is true in the world state. This calls for a binding for uGx14. Thus an axiom test is performed. The atom '((before elt2)(items-of give1 uGx14))', in sequent [P16d], is matched against the atoms in the world state, in which the timed expression,

TABLE 4.VIII: Deductions from [P16a] and [P16c].

Name	Sequents	Rules
[P17a]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)][*], ((before (starting-time-of give1))(owns seller-2 car-1))))[*], ((before (starting-time-of give1))(owns seller-2 uGx14))), <service-exp> → ...,(CREATE(1) ((EXISTS x CAR)...)) ;	
[P17b]	((before elt3)(**value-of** (*function* give2))), ((before elt4)(**value-of** (*behavior* give2))), [(before elt1)(*function* give1)], ((before (starting-time-of give1))(owns seller-2 uGx14)))), ((before (starting-time-of give1))(owns seller-2 car-1)))), → ...,(CREATE(1) ((EXISTS x CAR)...)), (uGx15 ∈ ((before elt2) SERVICE) ;	 [→ CRR-E] [CREL-1]

((during [(starting-time-of give1) (ending-time-of give1)])
(items-of give1 car-1)]

is currently true. The time term, '(before elt2)' will match with during-expression above, since '(before elt2)' intersects with the interval of the during-expression. This causes uGx14 to be bound to car-1. To simplify matters, let us suppose that the seller-2 does own car-1. Then [CREL-1] rule is applied resulting in the CREATE expression being replaced by its argument, as shown in sequents [P17a] and [P17b]. Incidentally, this also causes sequent [P16d] to become an axiom, as shown in the table.

The next logical step is to analyze the <service-exp> in the problem sequent [P17a]. If this is done it will lead to a dead end since no appropriate bindings for uGx15 (whose range is a SERVICE) exists in the world state: For all services, x, in the world state it is not known whether (items-of give1 x) is true or false: it is unknown. Thus this line of analysis will be dropped. Let us choose the *function* of give1 for analysis now. The expansion for this *function* is the first expression in sequent [P14a]. A similar analysis of this expression will now cause the creation of '[(owns John car-1) AND (NOT(owns seller-2 car-1)]' to occur before elt1, i.e. before the time of successful termination of give1. To make matters simple let us suppose that these are created by simply asserting them into the world state.

The assertion of '(owns John car-1)' before elt1 (i.e. the time when give1 ends) will cause the CC at (status-of give1) to be evaluated, because '(status-of give1)' is dependent on '(owns John)', it is in $\mathbf{D}_{[\text{owns John}]}$. The evaluation of the CC at '(status-of give1)' will now cause this status to be set to successful by default.

At this point the '(CREATE(1) ...)' expression that is on the right side of all the problem sequents is already true in the world state, because John got the ownership of his car before he paid for it. Thus, if [CREL-1] rule is applied to the sequents in the frontier set after the application of [→ CRR-E] rule to remove the existential quantification, all the sequents will reduce to axioms. However, John has not yet paid the money for the car.

I could have prevented this situation by associating a CC with '(owns PERSON)' which prohibits ownership of an object unless it is paid for, i.e. the cost of the item had been already given to its previous owner. I did not do this here. This brings to focus a general problem in analyzing parallel actions: It is impossible to anticipate all the conditions that govern the successful completion of a set of parallel actions.

A point of view that has influenced the design of CK-LOG is that, knowledge specification *should not require a knowledge engineer to anticipate all possible ways in which knowledge units might interact.* To the knowledge engineer his perspectives during knowledge specification should be always local to each concept. The system should be able to take care of the interactions between the various concepts. The use of *parallel predicate flags* provides the necessary extra logical capability to respond to pathological situations like the one illustrated above, which might arise as a result of this.

Thus, at this point the presence of the common parallel predicate flags in the remaining unanalyzed expressions of sequent [P13] will now cause these expressions also to be expanded and assimilated into the world state. This will result in the successful completion of give2, thus solving the problem, under the hypothesis [HP4], namely that John did decide to buy this car.

The phenomenon pointed out above illustrates another characteristic of proofs in CK-LOG. During the proof process the world states associated with sequents in the deduction tree may change. As a result of changes made to a world state, W, associated with one sequent, Q, it is not just Q alone that might acquire a new world state, say

W′, many other sequents which had the same world state W associated with them may also acquire the new world state W′. Such a change may cause many sequents besides Q to reduce to axioms. Thus as the effects of actions change a world state, several problems in a deduction tree may get reduced to axioms. The reverse may also happen. A problem that had been previously reduced to an axiom might loose its axiom status as a result of acquiring the new world state W′. This phenomenon of interaction between the action calculus and proofs is unique to CK-LOG. It does not occur in conventional natural deduction proofs.

How should one view this proof discussed above? Does it constitute an execution of the '(CREATE (owns John car-1))' action? It does not because the time instants at which the actions occur have not been specified. They have been stated only relative to each other, and that they all occur before [1984 Oct 1]. The proof displayed above may however be used to generate a plan for John owning a car. This may be done by extracting from the proof the problem sequents ([P1], [P5], [P12]) and replacing the constants that appear in these sequents with the variables to which they were bound to. The resultant problems are shown below in table 4.IX. These three problems taken in sequence may now be viewed as a plan for solving the initial problem.

TABLE 4.IX: A Plan for Car Ownership by John.

Name	Sequents
[P1]	→ (CREATE ((EXISTS x CAR)([after (1984 Sept)][owns John x]))) ;
[P5]	→ (**R-assert** ((eGx1 ∈ CAR),(eGx2 ∈ SELLER),(sells eGx2 eGx1), ([before (1984 Oct 1)][eGx3 ∈ BUY]), ([before (1984 Oct 1)][seller-of eGx3 eGx2]), ([before (1984 Oct 1)][buyer-of eGx3 John]), ([before (1984 Oct 1)][items-of eGx3 eGx1])), (CREATE(1) ((EXISTS x CAR)...)) ;
[P12]	(eIc1 ∈ (cost-of eGx2)), (**L-assert** [((between (*interval-of* eGx3))(GIVE eGx2 John eGx1)), ((between (*interval-of* eGx3))(GIVE John eGx2 (price-of eIc1)))]) → ..., (CREATE(1) ((EXISTS x CAR)...)) ;

This plan has a critical time element. It is valid only if it occurs before [1984 Oct 1]. At the time of execution of this plan John will be asked to select the car he wants and buy it from the dealer he chooses to buy it from. John would also be forced to select the time instants at which he wishes to perform the various actions. These time instants should, of course, satisfy the time conditions given in the plan. Thus there are clearly two distinct modes of action interpretation in CK-LOG: Plan formation mode and plan execution mode.

Forming a plan by eliminating some of the details in a proof tree serves two purposes: It makes the plan statement brief, and it also allows for possible variations in a plan at the time of its execution. The way a problem was solved at the time of plan generation may not exactly coincide with the way it gets solved at the time of plan execution. By representing in a plan only the important and critical problems extracted from a proof tree, and dropping off the details of problem solution, one allows for the possibility that these problems might get solved differently at the time of plan execution. The problem details and varible bindings presented in a plan would represent the details that one ought to expect at the time of its execution. During execution if one or more of the problems stated in a plan does not occur, or occurs in a different form, then this will indicate a departure from the plan, calling for either plan revision or action modification. Thus plans represented in CK-LOG may be used to guide plan execution.

Two questions arise about extraction of a plan from a proof: In general, which sequents should one extract from a proof in order to form a plan for the proof? One possible guiding principle is that problem sequents where the world state is modified should be included in a plan, and sequents where no world state change occurs may be omitted. This is not a sufficient characterization of the plan extraction process, because there are numerous exceptions to this principle. Could one always choose to remove from a plan the bindings given to the variables in the proof? In the Naval Operational Planning domain situations do occur where one wants to keep some of the bindings in the plan (such as for example the bindings given to the forces that are going to be used in a military action). The bindings that occur at time instants in a future time (with respect to the planning time) could be removed in many cases, but not always so. Thus, at the moment I have no definitive answers.

In the operational planning problem the concept of an operational plan, called OPPLAN, is rather clearly defined. Thus one could extract from a proof the information needed to complete an OPPLAN specification. I believe, this is indicative of the general situation. One has to define what one means by a plan in order to extract a plan from a proof. If the *concept* of a PLAN is available, then a proof like the one above may be viewed as a plan generation process with respect to the given definition of PLAN. The plan corresponding to a proof may then be extracted from the proof. The *design* of PLAN will specify this extraction process. This PLAN *design* could depend on the actions invoked in a proof and perhaps also on the rules used in a proof, or more generally on the structure of a proof tree since proof trees are available to the system as objects. The definition of the concept of PLAN is by itself an interesting problem for further investigation.

In the context of CK-LOG's plan execution mode, a plan may be viewed as a specification of a program to accomplish a goal. The problems and variable bindings stated in the plan specify the problems and bindings that one should anticipate to encounter in trying to achieve the goal of the plan. Any departure from this will indicate a need for plan revision, which may then be done at the time of plan execution. The nature of this plan revision problem thus depends on the concept of the plan itself.

Besides plan generation, plan execution and plan revision, the TPS may also be of course used to answer questions about a plan, or about a given course of action. One may pose to TPS problems in which the objective is to show that if certain courses of actions are followed in certain situations, then certain consequences would follow. For each action and each plan of action CK-LOG has the full capability to analyze all aspects of the action, the plan and its execution. This is the most significant difference between CK-LOG's mode of describing actions, and actions described through *procedural nets* [Wilkins 1982], or through *procedural attachments* [Bobrow, 1977].

The most significant points to be noted in the above action calculus are the following two:

1. The action calculus functions in an environment of incomplete knowledge about world states, and
2. Action definitions do not require *frame axioms* [McCarthy 1969].

The *frame problem* pointed out by McCarthy simply does not arise in CK-LOG's action calculus. This is because of the way the TPS and TMS interact. During the theorem proving process TPS uses TMS to build partial models of world states. At every point in the proof generation process the world states associated with the problem sequents delimit the deductions that TPS could make. When an action occurs, the world state is changed selectively to reflect only the changes introduced by the action. The rest remain unchanged. The use of modal operators in CK-LOG makes this possible. Thus there is no *frame problem* in CK-LOG.

The action calculus in CK-LOG provides a powerful and compelling alternative to the *situation calculus* of McCarthy.

The knowledge representation scheme presented here is rather a complex one. How easy is it to describe knowledge in this manner? To this author it seems quite easy, but there is no doubt that the knowledge specification can itself be a formidable task for large domains†. This is to be expected. Even in situations where one thinks one knows a situation well, the description of the situation is often not easy in any language, formal or natural. It requires training and practice. We have been involved in codifying knowledge in various domains for centuries now. But in this process we have been communicating only with other humans, who with difficulty and practice can at times master the knowledge. The trend to communicate in a like manner with machines has not even begun yet. I believe that with better understanding, more practice and better implementations of CK-LOG like systems one would be able to formulate better both the pragrmatic and logical problems involved in *knowledge engineering*. I would like to discuss in the next section some preliminary ideas on the nature of this knowledge that one describes to CK-LOG and the criteria that one might use to design a representation for it.

---

† I do not yet have enough experience with the use of CK-LOG to comment on the difficulties and problems that the use of CK-LOG poses. I have often encountered trouble in the use of ACHIEVE, CREATE and DESTROY operators. The trouble had been always with the following kind of difficulty. In the semantics given to CREATE operator,

$$|(\text{CREATE } (\text{X IMPLIES Y}))| \equiv |(\text{DESTROY X}) \text{ OR } (\text{CREATE Y})|,$$

which is, of course, the way it should be. Often, when one defines *functions* and *behaviors*, one may encounter expressions like, '((EVERY x .)(CREATE ...))'. If one now invokes a *function* through '(ACHIEVE (*function* ...))' and this function had the above universally quantified expression then the interpretation of this ACHIEVE will not result in the intended interpretation for the function. In cases like these one has to invoke the *function* using (value-of (*function* ...))'. In general,

$$|\text{X IMPLIES (operator Y)}| \neq |\text{operator (X IMPLIES Y)}|.$$

I have repeatedly gotten into trouble because of this. One has to be careful. It is generally safer to set goals '← (value-of X) ;', than goals '← (ACHIEVE X)'. If the value of X is an operator free expression then ACHIEVE should be used. If it is not then value-of should be used. However, this assumes that at the time the value of X was defined its mode of call should be anticipated. This feature is quite idiosyncratic to the current set of ACHIEVE and value-of definitions used in CK-LOG. There is no intrinsic reason why this could not be changed.

5. The Logic of Frames in MDS.

My objective here is to enunciate a principle of design for creating frame representations of knowledge, that are appropriate for systems like CK-LOG. In the discussions below I will refer to the knowledge K[U] without its *function, behavior, analysis and design* aspects as the *frame theory of* U, and use the notation **F[U]** to denote this theory. The theory **F[U]** will thus consist of all the dimensions and the CC's associated with the dimensions. This is the *theory of statics,* for the universe **U**, namely the theory that characterizes all the action free world states of U (one may think of them as snap shots of world states at given instants of time). The specification of the aspects of knowledge other than the *structures* then provides the *theory of dynamics* for the universe. The discussion below is concerned only with the theory of statics. There is not much known yet about the theory of dynamics. There is definitely a need to get results on the nature of this theory of dynamics similar to the results discussed below for the theory of statics.

One may view logical languages like TML and DL as offering certain facilities to distinguish between distinct constants in an universe through distinct descriptions in the language. I will later make this notion precise. Roughly speaking one might say that the kinds of distinctions that one is able to recognize using a language delimits the kinds of problems and solutions that one can express and solve using the language and its interpretations. Given the power of a first order language, the principal emphasis in the design of frame theories falls on *the building of the right kinds of expressive facilities that facilitate efficient discrimination of constants in a universe.* I would like to formalize this notion and claim that it is an useful notion. It can be the *guiding principle* for designing *frame theories.*

Let us say that the *resolving power* of a first order language without the equality symbol** L, is expressed by the equivalence classes of constants distinguished by the language, i.e. c and d belong to different equivalence classes only if there is a description D(c) of c and a description D(d) of d such that in some world state D(c) is true and D(d) is false. If we have a complete and consistent theorem proving system, and a *complete specification* of the knowledge **F[U]** (this notion needs to be made precise), and if L

** This a technicality that is intended to prevent two constants from being distinguished just from their names.

resolves c and d then the system would be able to prove that c is not equivalent to d in L. Indeed, if the theorem proving system is complete and consistent then one may use the expressive power (the resolving power) of the language L to characterize the concept of *completeness* of F[U]: It is complete iff for every c not equivalent to d in L, the theorem proving system is able to prove using F[U] that c is not equivalent to d. The problem of finding such a F[U] is similar to the problem of finding a complete set of axioms for a formal system, a difficult problem in any non-trivial system. Principles of design that give local guidance to the design of such a system are thus valuable principles. The *locality principle* first mentioned in section 2 is one such principle. This principle and the formal notion of a *frame* are defined in this section.

Let me introduce some definitions. Let L be the domain language without the equality symbol. It does not contain modal expressions. Let D be the set of all *constants* in the universe U, each with a unique and distinct name. I will assume throughout that all the world state models, U_t , i.e. sets of sentences (closed formulas) in L which are true in a given interpretation, are contradiction free. They may, however, be incomplete in the sense there may be sentences whose truth value is unknown in the models. Let U be the set of all such world state models in the universe. I will use the notation U_t[c d] to denote a world state that contains constants c and d, and given U_t[c d] I will use the notation U_t[d c] to denote the world state obtained from U_t[c d] by interchanging the names of c and d (i.e. the names are switched in every sentence of U_t[c d]). Then,

[D1]. *Distinguishability.* Constants c and d are distinct only if there is a finite U_t[c d] in U such that U_t[d c] is not in U.

For example, if '(father-of c d)' appears in a world state U_t[c d] in U then clearly U_t[d c] is not in U.

[D2]. *Subset* L(c) *of* L: L(c) is a subset of L defined by the constant c,

$$L(c) = \{S(x) \mid ((\text{EXISTS } U_t\ U)(\text{ISTRUE } S(c)\ U_t))\},$$

where S(x) is an arbitrary formula of L with one free variable x.

[D3]. *L-equivalence*: [(c $=_L$ d) IFF L(c) = L(d)].

[D4]. *Resolving Power of* **L**: This the set of all L-equivalence classes of **D**, denoted by **D/L**.

Theorem 1: Two constants c and d are *indistinguishable* (i.e. not distinguishable as per [D1]) if and only if (c $=_L$ d).

Proof: L-equivalence implies that every logical condition that c satisfies in some world state is also satisfied by d in some world state. The essential part of the proof is to show that if c and d are L-equivalent then then they satisfy the identical sentences in every world state in which they are present.

Suppose c is L-equivalent to d and there was a world state U_t in which some formula S(x) is true for x = c, and false for x = d. Then ((NOT S(d)) AND S(x)) is in L(c) but not in L(d), a contradiction. Thus in every world state in which c and d exist and S(c) is true, S(d) is also true. Suppose S(c) was true in U_t and S(d) was unknown. Then there is a consistent extension of U_t in which S(d) is no longer unknown. In this extension S(c) should have the same truth value as S(d). In this case U_t[c d] and U_t[d c] are both in U. Hence c and d are indistinguishable.

If c and d are indistinguishable then for every U_t[c d] in U, U_t[d c] is also in U. Hence it follows that c is L-equivalent to d.

Corollary 1.1: If (c $=_L$ d) then for every relation name, r, in every world state in which both c and d exist, either (r c) = (r d), or there is an extension of this world state in which (r c) = (r d).

Corollary 1.2: If (NOT (c $=_L$ d)) then there is a U_t and a sentence of the form

S1(c d) = [S(c, d) AND P]

which is true in U_t for which the corresponding sentence

S1(d c) = [S(d, c) AND P]

is false in every complete world state in U.

Proof: P here is a sentence that would depend on c and d. Since c is not L-equivalent to d, there is a finite world state, U_t[cd] in which for some relation name, r, (r c) $\neq$ (r d), and U_t[d c] is not in U. Let P be the conjunction of all the sentences in U_t[c d] in which neither c nor d appear. Let S(c, d) be the conjunction of all the sentences in U_t[c d] in which c or d appear. This conjunction will use all literals with relation names, r, for which (r c) $\not\approx$ (r d). Since U_t[d c] is not in **U**, the sentence S1(d, c) above has to be false in every world state of **U**.

Thus if c and d are L-equivalent then clearly there is no way by which one can create descriptions in **L** that distinguish them. In this sense **D/L** defines the resolving power of L. The equality relation is excluded from **L** because otherwise every equivalence class in **D/L** will be trivially a singleton class, since the formula (x = c) is true when x is c and false when x is d.

Let $\mathbf{U}_S$ be a subset of **U** such that for every pair of constants, c and d, if c and d are distinguishable in **U**, then they are distinguishable also in $\mathbf{U}_S$.

[D5]. *Adequacy of* **F**[**U**]: **F**[**U**] is *adequate* if every world state in $\mathbf{U}_S$ is a model of this theory, and a world state not in **U** is not a model of the theory.

Thus $\mathbf{F}_S \subset$ models of **F**[**U**] $\subset$ **U**. Notice that the *adequacy* notion is thus a weaker notion than the notion of completeness and consistency of a theory. If **T** is a complete and consistent theory of **U** in the language **L**, then the theorems of **F**[**U**] could be a superset of the theorems of **T**. These additional theorems of **F**[**U**] will be hopefully such that they help distinguishability at the expense of loosing ones ability to distinguish all distinct constants in all possible ways. The following theorem fixes the relationship between the deductive closure, Θ, of an adequate **F**[**U**] and L(c) for c in **D**.

Theorem 2: [(((EVERY x)S(x)) $\in \Theta$) IFF ((EVERY y)(S(x) $\in$ L(y)))]
[(((EXISTS x)S(x)) $\in \Theta$) IMPLIES ((EXISTS y)(S(x) $\in$ L(y)))]

Proof follows from the definitions of *adequacy* and L(c). The resolving power of **L** is related to the completeness and consistency of the theorem proving system (TPS) by the following definition:

[D6]. **F**[**U**] *resolves a domain* **D** if for every pair of constants c and d in **D** that are distinguishable there is a formula S(x) such that

$[(S(x) \in L(c))$ AND (NOT $(S(x) \in L(d)))]$ and

$F[U] \rightarrow [S(c)$ AND (NOT $S(d))]$; is valid.

Notice that not all the formulae that distinguish c and d need be provable. Also, this assumes that F[U] somehow knows about all the constants in D. Thus a theory F[U] resolves D if it can be used to prove the distinguishability of all pairs of constants that are not L-equivalent:

Theorem 3: If F[U] is *adequate* then it resolves the domain D.

Proof: If a theory is adequate then for two distinguishable constants c and d, there is a finite model $U_t[c\ d]$, for which its corresponding $U_t[d\ c]$ is not a model. Clearly then, by corollary 1.2 there is a sentence S1(d, c) which is false in every model. Now construct the sentence (NOT S2(d, c)) by uniformly replacing every constant in (NOT S1(d, c)) other than c and d by new variables which are all existentially quantified. Then (NOT S2(d, c)) will have the form,

(NOT S2(d, c)) = [(EXISTS x1)(EXISTS x2)...(EXISTS xn)(NOT (S1(d, c) AND P′))],

where P′ is obtained from P by uniformly replacing all the constants other than c and d by the variables x1, x2, ..., xn, respectively. This sentence is true in every world state. Hence the following sentence is a logical consequence, and a theorem in the theory:

S3 = ((EXISTS x)(EXISTS y)(NOT S2(d c))).

Also, by corollary 1.2 S2(c, d)) is true in U_t . Then the following sentence provably distinguishes the constants c and d:

S(y) = [S2(d y) AND S2(y d)],

because S(y) is in L(c) but not in L(d). Hence the theory resolves D.

Our objective in designing frame theories is to get theories that are *adequate* in this sense, namely using the theory we should be able to distinguish all distinguishable (non L-equivalent) constants. Among all such complete and consistent theories, for a system like CK-LOG, we need theories that satisfy an additional condition. I have called this condition the *locality condition*. I mentioned in section 2 that the locality condition guarantees the retrieval from the world state of all the conditions that are relevant to a given literal asserted into the world state. I would like to establish here the truth of this statement and also show that theories satisfying the locality condition are *adequate* theories. To do this I need to define the notion of a *frame*.

5.1. Frames and Dimensions.

Let the P(c) be the *predicate signature* of c defined as follows:

[D7]. $\mathbf{P}(c) = \{r \mid ((\text{EXISTS } \mathbf{U}_t\ \mathbf{U})(\text{EXISTS } y)(\text{ISTRUE } (r\ c\ y)\ \mathbf{U}_t))\}$.
where r is a relation name*.

Clearly, if $\mathbf{P}(c) \neq \mathbf{P}(d)$ then c and d are distinguishable. The concept of P-equivalence may now be defined as,

[D8]. *P-equivalence*: Two constants c and d are P-equivalent if and only if $\mathbf{P}(c) = \mathbf{P}(d)$.

Clearly, L-equivalent classes are refinements of P-equivalent classes. Since the set of relation names in **L** is a finite set, there are only a finite number of P-equivalent classes in **D**. Let,

$$\mathbf{X} = \{X_1, X_2, ..., X_N\}$$

be the set of all P-equivalent classes in **D**. For a class, X, let the predicate signature, **P**(X), of X be the same as **P**(c) for any c in X. One may now define a dimension of X as,

* I will assume that only binary relation names are used in **L**. Clearly all n-ary predicates can be described using the appropriate number of binary relations.

[D9]. *Dimension*: (r X Y) is a dimension of X only if there exists a c in X, a world state U_t , and a d in Y such that (ISTRUE (r c d) U_t) = T.

This is essentially the same definition given in section 3, but now we have a model theoretic understanding of what this concept X is. Every concept, X, used in a knowledge representation system like CK-LOG is thus a member of **X**. One may say that Z is a generalization of X if P(Z) is a subset of P(X). The inheritance rules that one uses over this generalization hierarchy appear to be sheer inventions. There is no logical reason why they should work. But if one assumes that one uses the same relation name, r, in anchors (r X) and (r Y) for different X and Y, only because the properties of (r X) are in some sense similar to the properties of (r Y) then the use of inheritance rules makes sense. Most of the time the properties imposed by the inheritance rules coincide with the nature of things in U. However, exceptions do always occur and all frame systems provide facilities to define such exceptions.

In defining frame theories one is attempting to identify the equivalence classes X defined above. A *frame*, X, is simply the set of all dimensions of the class X. If one can identify **X** for an universe U then the dimensions of the classes in **X** are precisely the frames that one would want to use in the representation of the knowledge of the universe U. We go about doing this by choosing an appropriate set of relations, and defining the dimensions for them. It is fortunate that frames defined in this manner coincide with the frames defined by the classes in **X**. It must have something to do with the way we use relations in languages.

One may now define the distinguishability criterion in terms of the relation paths defined by the frames. Clearly, if there is a relation path, p of length greater than 1, such that for two constants c and d,

$$(\text{NOT } [(c \ \mathbf{p} \ d) \text{ IFF } (d \ \mathbf{p} \ c)])$$

is true in a world state then c and d are distinguishable. If on the contrary no such relation path exists in any world state for c and d, then c and d are L-equivalent. The following is a general statement of this condition for a *distinguishing relation path*, **p**:

[D10]. *Distinguishing relation path:* p_{cd} is a distinguishing relation path for constants c and d, if ((EXISTS z)[(c p_{cd} z) AND (NOT (d p_{cd} z))]) is true in some world state.

I will refer to the condition in [D10] as a *distinguishing condition* for c and d. In frame systems the constants that belong to different classes X and Y are already distinguishable by their class memberships. However, to distinguish the distinct constants that belong to the same class, X, one needs additional facilities besides the facility to define dimensions. In frame systems like MDS, this additional facility is provided by the definition of the CC's.

In general one may think of a ccexp at each anchor (r X) as a potentially infinite conjunction of all the distinguishing conditions associated with pairs of instances c and d of X, all of whose *distinguishing relation paths* have the form (r . p), i.e. produced by the concatenation of the relation name r to p. However, since we have only a finite number of relation names, the regularities which must inevitably exist among the conditions that occur in this infinite conjunction, enable us to state the conditions using finite sentences. In such a CC for every distinct pair of instances c and d of X, every distinguishing path p_{cd} starting with, r, will occur implicitly in the ccexp.

The assignment of a value for (r c) will cause a contradiction only if for some c1 and d1 in the world state (not necessarily instances of X) this causes one of the distinguishing conditions for c1 and d1 to be violated. CC's of this kind will satisfy the *locality principle* stated in section 2 for the following reason.

Suppose a change δ occurred at (r c), and as a result of this change a potential contradiction may arise at another anchor (r′ d), where d is an instance of Z. Then in the ccexp at the anchor [r′ Z], there should be a distinguishing condition which is likely to become false as a result of the change at (r c). Let

[(EXISTS w)(d q w) AND (NOT (d1 q w))])

be this condition. This condition may become false as a result of the change at (r c) only if the truth of this condition depended on the change δ that occurred at (r c). For this to happen there should be a prefix of q, of the form (q′ . (r)) , such that

[(d q′ c) OR (d1 q′ c)],

is true, or was true in the world state that existed before the change at (r c). Or there should be prefix of q, of the form (q′ . (cr)), where (r, cr) is the converse relation name pair, such that for some constant b in δ,

$$[(d\ q'\ b) \text{ OR } (d1\ q'\ b)]$$

is true or was true in the world state that existed before the change at (r c). In either case, since in our system for every relation its converse is also in the model, there is a relation path between either c and d or between c and d1. Hence the locality condition is satisfied.

Thus every constant in a world state that can potentially cause a contradiction as a result of a change at (r c) will be related to c via a relation path. Thus all the affected constants are in a sense local to c, i.e. reachable from c. Hence the name *locality condition.* It seems only reasonable that for each anchor (r c) there should be a frame surrounding it which includes in it only the anchors that are affected by the changes at (r c). This is precisely what *frame systems* allows one to do. When performing a contradiction check this will force the system to check every potential dependency that exist in a world state, thus assuring that TMS would return to TPS all the logical conditions relevant to a given set of changes.

A second consequence of this view of CC's is the following. For any z, (r′ z) would be in the *dependency set* $D_{[r\ c]}$ iff there is a relation path q such that for some constant b in δ, (b q z) was true either in the new world state or in the old world state. By analyzing the CC's one can identify these relation paths q and use them to define the filters mentioned earlier in section 3.

These observations should provide some guidance to a knowledge engineer in writing the CC's and in selecting the dimensions for a domain. They also hopefully set some preliminary (and admittedly very rudimentary) criteria for a possible theory formation system which seeks to identify the classes X in a domain, discover the dimensions and write CC's for them. I believe that a deeper understanding of CK-LOG like systems will one day enable us to build such learning systems.

6. Concluding Remarks.

I have here presented a logical calculus for processing knowledge, CK-LOG, that integrates knowledge representation using *frames* with problem solving using *theorem proving*. The organization and operation of CK-LOG incorporates the following novel features:

1. Combines a theorem proving system based on natural deduction with a frame based knowledge representation system;
2. Extends the essentially structural notion of *frames* to the more general notion of *concepts*;
3. Introduces a new termination algorithm, called the *mating algorithm*, for the theorem proving process;
4. Extends the standard notions of problem solving through theorem proving to a logical system that uses *modal operators*;
5. Shows how specialized reasoning facilities and inference rules may be incorporated in a theorem proving system to represent and reason about actions and their time dependencies;
6. Shows how models may be used in such a theorem proving system to represent and reason about *possible worlds* associated with the modal expressions;
7. Shows how a modeling system based on three valued logic may be used to represent *defaults*, or identify information that is needed for the solution of a problem but is unknown in the world state;
8. The analysis in section 5 provides the beginnings of a deeper logical understanding of the nature of frame systems and their design.

CK-LOG offers a powerful and compelling alternative to the *situation calculus* [McCarthy 1969] approach of representing and reasoning about actions. It is possible to implement CK-LOG in a manner that it has the meta-level reasoning abilities to reason about its own problem solving processes. The current implementation of CK-LOG does not have this capability. Being the first implementation of its kind there is room for much improvement both in program organization and the implementation technology. Hopefully such improvements will be forthcoming in the future. They are vital to achieve the performance that one has come to expect out of frame systems. Also there is need to gain experience on the use of CK-LOG as a knowledge processing system to build 'expert systems' in complex domains. I am now using the building of OPPLAN-CONSULTANT [Srinivasan 1984] as an experimental vehicle. This planning domain poses several challenging problems. It is this challenge that helped crystallize the many innovations in CK-LOG.

Besides the problems in improving efficiency and implementation techniques mentioned above, CK-LOG poses several fundamental logical problems which need further study. Most of these are concerned with the monitoring of multiple actions by the TPS/TMS systems. We have to understand better the way *action predicates* mediate action monitoring between the TPS and TMS, and develop some guidelines for the specification of action *functions* and *behaviors*. The few examples of these discussed in [Srinivasan 1984] are not enough to inspire any general patterns of organization. There is also room for some changes in the inference rules themselves, those that were specified for the modal operators. Here again more experience is needed with the use of the rules.

I did not analyze the time function, tmxn that was introduced in section 2.9. It would be useful to describe tmxn itself in terms of a set of inference rules. This is not for the sake of better system performance, but for better analysis of its properties. It should not be difficult to do this. I have chosen to represent and reason about time in terms of *time instants*, instead of *time intervals* as Allen did [Allen 1984]. I believe, the choice here is just one of style and personal preferences. I do not think *interval* representation has any special advantages to offer, either from implementation point of view or from the point of view of logical elegance.

There is also a whole set of problems pertaining to the use of CK-LOG as a *knowledge based learning system*. I indicated at a few places in the body of the paper that CK-LOG does have a potential for being used as a learning system. Again, to understand the problems here some experimental work is necessary.

I have presented in this paper an informal discussion of the *static theory* of CK-LOG. I have not said enough about its *dynamic theory*: its *function, behavior* and *analysis* as a knowledge based problem solving system. I have not adequately dicussed the dynamics of interactions between TMS and TPS, and how through such interactions and through the use of *action predicates*, CK-LOG might analyze the effects of on ongoing actions on a world state. A preliminary discussion of some aspects of these appear in [Srinivasan 1984]. Much more experimental work is needed in this area.

Let me conclude this paper with a short historical note on MDS and the evolution of ideas that led to CK-LOG. MDS was first presented as a knowledge representation system in 1973 IJCAI at Stanford, California. At that time an initial implementation of MDS, called TEMPEST (a template establishment system) was available. What came to be called *frames*, after Minsky's paper [Minsky 1975] were then called *templates* in MDS. I did not then have good ideas on how one might use the knowledge represented in MDS to state and solve problems. I thought mostly in terms of procedures written in a language which was then called the *designer* [Srinivasan. 1973a] but it was never implemented. The ideas proposed in MDS were first implemented into a working system by Sridharan [Sridharan 1978]. This system was called AIMDS (Action Interpretation Meta Description System). It used the modeling system proposed in MDS, but used *procedural attachments* to solve problems using its model building capabilities.

The possibility of using the natural deduction system of Kanger in MDS was first explored in 1973. But, because of several interruptions in my work this idea was not pursued vigorously for a long time, until I started work on the Naval Operational Planning problem in 1982. This led to the results presented in this report. I am thankful to the Navy Center for Applied Research in Artificial Intelligence for providing me this opportunity.

7. References.

Allen, J.F. [1984] "Towards a General Theory of Action and Time", *AI Jl.*, Vol 23, No. 2, July. pp. 123-154.

Bobrow, D.G, [1977] Kaplan, R.M., Kay, M., Norman, D.A., Thompson, H., and Winograd, T., "GUS, a Frame-Driven Dialog System", *AI Jl.*, Vol 8, 2, pp 155-173.

Bowen, K.A. [1982] "Programming with full first-order logic," MI-10, (Eds.) Hayes, J.E., Michie, D., and Pao, Y.H., Ellis Horwood Ltd., Publishers, Chichester, and John Wiley & Sons, New York., pp 421-440.

Gentzen, G. [1935] "Investigations into Logical Deduction," *The Collected Papers of Gerhard Gentzen*, pp. 68-131, Amsterdam, North-Holland.

Goldstein, I.P., and Roberts, R.B. [1979] *Using Frames in Scheduling*, in *AI-MIT*, vol. 1, pp 251-284.

Irwing, J. & Srinivasan, C.V. [1975] *Description of CASNET in MDS*, RUCBM-TR-51, Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903.

Kanger, S. [1963] A simplified proof method for elementary logic, *Computer Programming and Formal Systems*, pp 87-94, [ed. Braffort, P. and Hirshberg, D.] Amsterdam: North Holland.

Kowalski, R. [1979] *Logic for Problem Solving*, Amsterdam & New York: North Holland Publishing Co.

Lenat, D.B [1976] *AM: An Artificial Intelligence to Discovery in Mathematics as Heuristic Search*, Ph.D. Thesis, AIM-286, STAN-CS-76-570, Stanford University, AI Lab., Stanford CA.

Levesque, H.J. [1984] "Foundations of a Functional Approach to Knowledge Representation," AI Journal, Volume 23, Number 2, July 1984, pp 155-212.

McCarthy, J. and Hayes, P.J. [1969] "Some Philosophical Problems from the Standpoint of Artificial Intelligence", MI 4, pp 463-502.

Minsky, M. [1975] "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P.H. Winston (Ed.), McGraw-Hill, New York, pp 211-277.

Robinson, J.A. [1965] A machine-oriented logic based on the resolution principle. *J. Ass. Compt. Mach.*, 12, 227-234.

Schmolze, J.G & Brachman R.J. [1982] *Proceedings of the 1981 KL-one Workshop*, Bolt Beranek and Newman Inc., Report No. 4842. Prepared for Advanced Research Projects Agency.

Sridharan, N.S. [1978] *AIMDS User Manual - Version 2*, CBM-TR-89, Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903.

Srinivasan, C.V. [1973a] "Architecture of Coherent Information System: A General Problem Solving System," *IJCAI-3*, pp 618-628. Republished in 1976 in *IEEE Trans. on Computers*, vol. C-25, 4, pp 390-402.

Srinivasan, C.V. [1973b] *Programming over a Knowledge Base, the Basis for Automatic Programming*, SOSAP-TR-5, Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903.

Srinivasan, C.V. [1977] *The Meta Description System: A System to Generate Intelligent Information Systems, Part I, The Model Space*, SOSAP-TR-20A, Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903.

Srinivasan, C.V. [1983] *Design for OPPLAN-CONSULTANT, An Expert System for Naval Operational Planning*, Final Research Report on work supported by ONR grant 4330/NOO14-82-C-2126, Department of Computer Science, Rutgers University, New Brunswick, N.J. 08903.

Srinivasan, C.V. [1984] *The Use of CK-LOG for Knowledge Representation and Problem Solving in OPPLAN-CONSULTANT: An Expert System for Naval Operational Planning*, Available in Manuscript form from the author.

Stefik, M. [1979] "An Examination of a frame-structured representation system", In *IJCAI-6*, pp. 845-852.

Wilkins, [1982] "Domain Independent Planning Representation and Plan Generation," Technical Nore No 266, AI-Center, SRI-International, Ravenswood Ave., Menlo Park, CA 94025.